NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

KINEMATIC AND STABILITY MOTION LIMITS FOR A HEXAPOD WALKING MACHINE

by

Elizabeth Marie Dunton

March 1995

Thesis Co-Advisors: Robert B. McGhee
Michael J. Zyda

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)

2. REPORT DATE
March 1995

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
Kinematic and Stability Motion Limits for a Hexapod Walking Machine

5. FUNDING NUMBERS

6. AUTHOR(S)
Dunton, Elizabeth Marie

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
The major problem addressed by this research is to investigate and implement the basic concepts necessary to lay the groundwork for efficient forms of motion planning, motion control, and gait algorithms with respect to hexapod walking machines.

Specifically, the approach taken was to develop and implement the concepts of a stability margin and a joint space motion margin on an object-oriented representation of the Aquarobot. The model was generated in Franz Common Lisp and simulated via Allegro Common Windows. A method by which distance computations can be calculated and applied to the center of mass and triangular support pattern of a walking machine to determine the stability margin is introduced. Inverse kinematics and joint limits are utilized to ascertain the joint space motion margin of the model. Response to impending instability and the effect when a joint hits or approaches a joint kinematic limit on the motion of the hexapod walking machine by stopping the model is also addressed.

The results are as follows: the concepts of the joint space motion margin and the stability margin can be successfully implemented on a kinematic model and graphical simulation of a hexapod walking machine. These concepts contribute to future work in the area of more efficient free gait algorithms, specifically asynchronous gait algorithms.

14. SUBJECT TERMS
joint space motion margin, stability margin, aquarobot, kinematics, robots, underwater walking robots, motion control

15. NUMBER OF PAGES
91

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

KINEMATIC AND STABILITY MOTION LIMITS FOR A HEXAPOD WALKING MACHINE

Elizabeth M. Dunton
Lieutenant, United States Navy
B.S.F.S., Georgetown University, 1985

Accesion For		
NTIS	CRA&I	☑
DTIC	TAB	☐
Unannounced		☐
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

March 1995

Author: ______________________
Elizabeth M. Dunton

Approved by: ______________________
Robert B. McGhee, Thesis Co-Advisor

____
Michael J. Zyda, Thesis Co-Advisor

____
Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

The major problem addressed by this research is to investigate and implement the basic concepts necessary to lay the groundwork for efficient forms of motion planning, motion control, and gait algorithms with respect to hexapod walking machines.

Specifically, the approach taken was to develop and implement the concepts of a stability margin and a joint space motion margin on an object-oriented representation of the Aquarobot. The model was generated in Franz Common Lisp and simulated via Allegro Common Windows. A method by which distance computations can be calculated and applied to the center of mass and triangular support pattern of a walking machine to determine the stability margin is introduced. Inverse kinematics and joint limits are utilized to ascertain the joint space motion margin of the model. Response to impending instability and the effect when a joint hits or approaches a joint kinematic limit on the motion of the hexapod walking machine by stopping the model is also addressed.

The results are as follows: the concepts of the joint space motion margin and the stability margin can be successfully implemented on a kinematic model and graphical simulation of a hexapod walking machine. These concepts contribute to future work in the area of more efficient free gait algorithms, specifically asynchronous gait algorithms.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENT

The author would like to thank Shirley Pratt for her guidance and patience in explaining many of the concepts behind the graphical simulation of Aquarobot and her help in debugging code. Thanks also go to Prof. Robert McGhee and Prof. Michael Zyda for their instruction and focus, without which this thesis would not have been possible. This work was supported, in part, by the National Science Foundation under Grant BCS-9109989 to the Naval Postgraduate School.

I. INTRODUCTION

A. AQUAROBOT

Developed under the auspices of the Port and Harbour Research Institute (PHRI) of Japan, the Aquarobot is a six-legged, "insect-type," underwater walking machine [Ref. 1]. It was built as a prototype for a possible alternative to human divers involved in the construction of seawalls along that nation's coast. Manpower costs required to operate Aquarobot as well as the low speeds associated with it, currently preclude Aquarobot as an economically viable option for the PHRI. The Naval Postgraduate School has created new control software to improve speed and efficiency, as well as a graphical simulator to test and evaluate this software before it is installed on the actual Aquarobot in Japan. Figure 1 shows Aquarobot [Ref. 2].

B. GOALS

The goal of this thesis is to investigate and implement various aspects of motion planning in the kinematic and graphical models. Specifically, the following issues will be addressed:

1. How can joint kinematic margins and the joint space motion margin of a hexapod walking machine be calculated and implemented?
2. How should the motion of a walking machine be affected when a joint hits or approaches the joint kinematic margin?
3. How can stability margins be calculated and implemented on a walking machine?
4. What should happen when the walking machine approaches a stability margin?

These questions are investigated and answered with respect to the Aquarobot. Experiments are conducted in Franz Common Lisp and simulated via Allegro Common Windows.

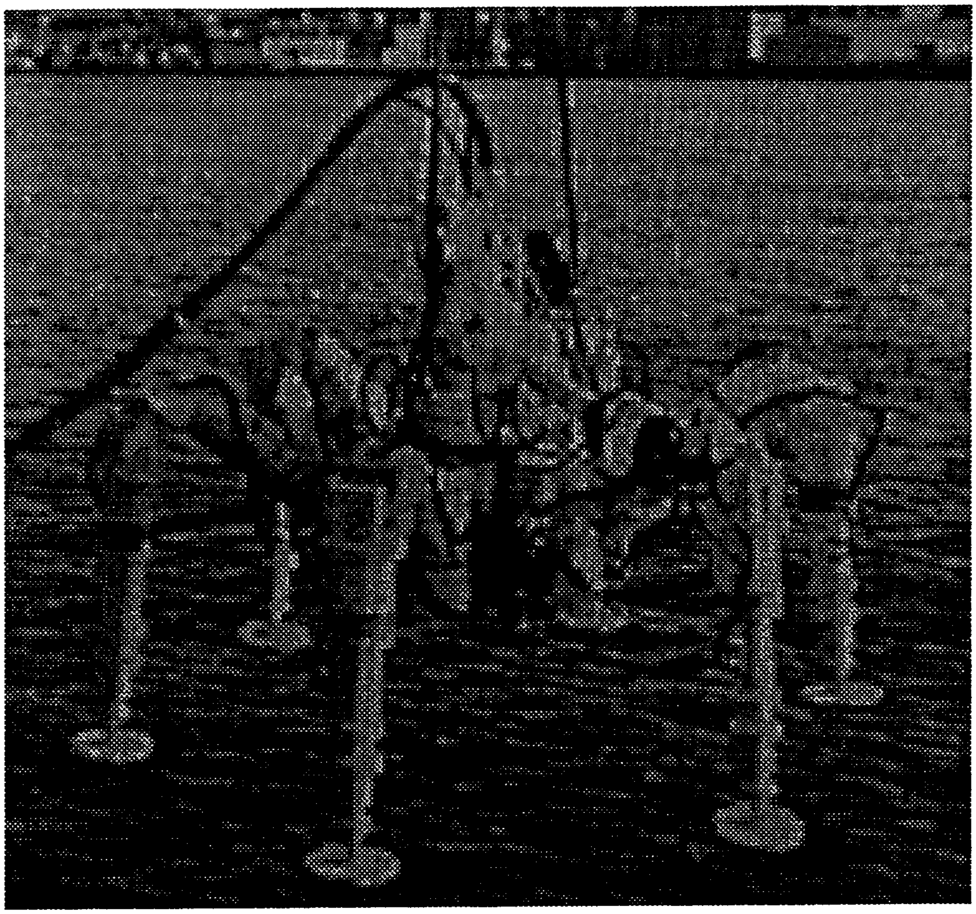


Figure 1: Aquarobot

C. ORGANIZATION

Chapter II reviews previous work regarding walking machines, kinematics and the Aquarobot vehicle. Chapter III discusses the kinematics model as well as the theory and implementation of joint limits and the joint space motion margin. Distance calculations and stability margin considerations are addressed in Chapter IV. Chapter V draws conclusions and offers suggestions for further research and investigation. Coded implementations of these concepts can be found in Appendix A. Appendix B contains a demonstration of a simulation illustrating concepts developed in this thesis.

II. SURVEY OF PREVIOUS WORK

A. WALKING MACHINES

Walking machines offer several possible advantages over standard wheeled-locomotion for off-road transportation. First of all, these machines have the potential to traverse rough and uneven terrain more quickly than those vehicles normally utilized. Secondly, they can improve energy efficiency. And finally, they appear to provide far greater flexibility and mobility over rugged surfaces. These advantages have spurred research and development into efficient and effective models for legged-locomotion. [Ref. 1]

1. General Electric Quadruped Transporter

One early effort was General Electric's Quadruped Transporter in 1965. It carried a human operator to provide sensor and neural control. The operator manipulated each of the four legs via four "position following, force-feedback" levers, alternatively known as *master-slave control.* Each lever incorporated three degrees of freedom - two at the hip and one at the knee (Figure 2 [Ref. 1]). The Quadruped Transporter first walked successfully in 1968 and proved that machines can, in fact, achieve animal-like mobility. Intended for use as a research vehicle, it proved extremely difficult to operate, rather fuel inefficient, and impractical to use. It did, however, encourage additional research into the field of legged locomotion. [Ref. 1]

2. OSU and MGU Hexapod Walking Machines

In 1977, Ohio State University (OSU) and Moscow University (MGU) each unveiled hexapod walking machines with the purpose of studying various previously developed terrain-adaptive coordination algorithms. Both machines sought to address several of the larger problems encountered by the Quadruped Transporter. While each continued to make use of a human operator, each incorporated the notion of *supervisory*

control. In accordance with this concept, the vehicle itself handled low level limb-control and coordination, while allowing the operator to concentrate on the general behavior and movement of the body of the vehicle with respect to speed and direction. [Ref. 1]

The OSU Hexapod was digitally controlled in an indoor laboratory environment. Each of the Hexapod's legs enjoyed three degrees of freedom (Figure 3 [Ref. 1]). The MGU Hexapod was controlled via a hybrid analog-digital computer. Both vehicles demonstrated the ability to execute complex maneuvers over rough terrain.

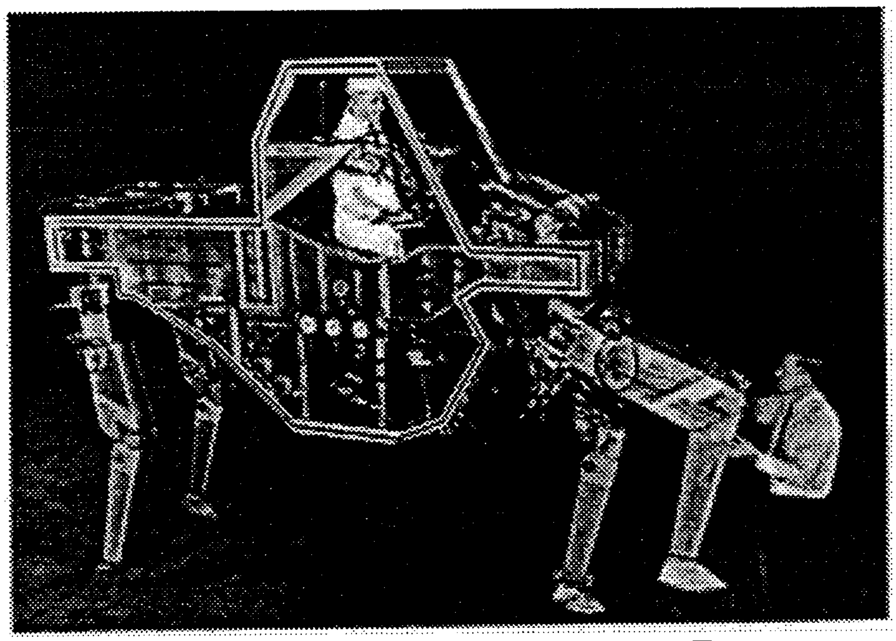


Figure 2: General Electric's Quadruped Transporter

3. OSU Adaptive Suspension Vehicle

In 1981, OSU began construction of the Adaptive Suspension Vehicle (ASV) [Ref. 1]. It also carried an operator who exercised supervisory control and utilized an onboard multiprocessor control computer to automate limb coordination and foot placement. It included a customized operating system as well as a self-contained, onboard power supply. Appropriate for outdoor use, it introduced an optical terrain scanner to generate a three

dimensional elevation map of the terrain ten meters in front of the vehicle. This information was then used to detect and select viable footholds. [Ref. 5]

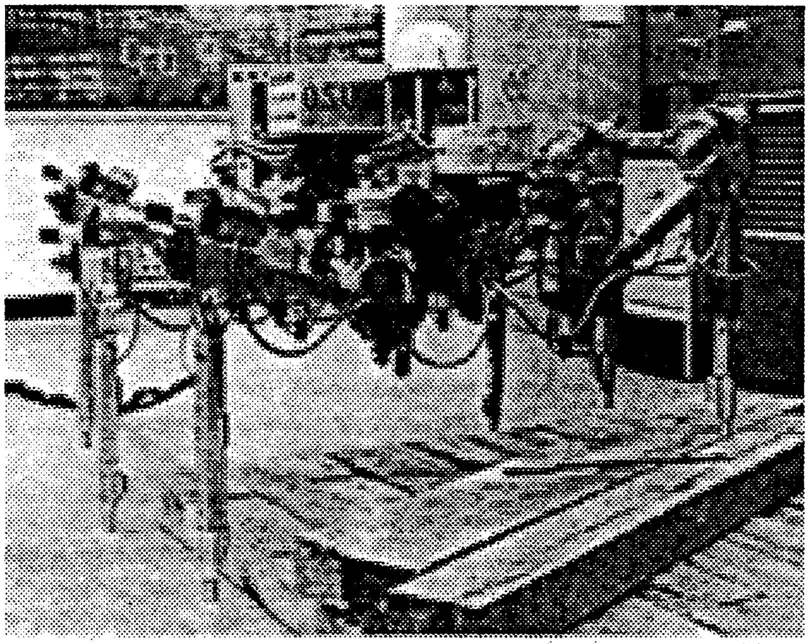


Figure 3: OSU Hexapod Vehicle

B. KINEMATIC MODELS/MOTION CONTROL

[Ref. 1] reviews the concept of supervisory control. The primary question is how to divide the control task between an operator and computer. There are two basic approaches: *top-down* versus a *bottom-up* approach.

A top-down strategy is best suited to terrain where every point accessible to a swinging leg is appropriate for load-bearing. Otherwise, a bottom-up control mode is preferable. In this case, successive footholds are chosen based on both reachability and load-bearing capability. Terrain is divided into *permitted* and *forbidden* cells by virtue of the reachability and load-bearing criteria. Research in this area has been slowed by the need for suitable sensors and for additional work in the area of terrain classification.

Inherent in motion coordination is the foot placement dilemma. While animals have the ability to *dynamically balance*, with just a few exceptions [Ref. 4], walking machines to date have had to rely on the concept of *static stability.* Under this system, legs are moved in such a way as to ensure that the vertical projection of the center of gravity remains inside the successive *support patterns* generated by those feet in contact with the ground. [Ref. 1]

[Ref. 5] takes the concept of static stability and develops an effective rule-based motion coordination system for an ASV-type walking machine on rough terrain. This experiment utilizes a *free gait* algorithm as opposed to the more conventional *tripod gait* routinely encountered in ASV-type walking machines. Tripod gaits are often ill-suited to rugged and uneven terrain because the area under a particular leg may be unable to support the machine at any given instant. Free gaits allow real-time optimization of leg placement and sequencing to ensure stability and maneuverability given difficult terrain conditions.

Leg control is based on the classification of foot motion into seven distinct states: ready, advance, descent, contact, support. lift, and return which define the "Leg Control Machine." Ready, descent, and support are asynchronous. Advance, contact, lift, and return are synchronous (Figure 4 [Ref. 5]). Leg movements are planned by the "Leg Plan Machine." The free gait coordinator sits at the top of the control hierarchy. It monitors the status of the walking machine and issues commands to control body and leg movements.

[Ref. 5] offers the following definitions:

1. A Foothold

A *foothold* is a point on the terrain. It can be assigned to a leg while the leg is in the air. Once the foot of a leg is placed on the terrain, this foothold becomes the *support point* of the leg.

2. The Support Pattern

The *support pattern* is the convex hull of the vertical projections of all support points onto a horizontal plane.

3. The Stability Margin

The *stability margin* of a support pattern is the shortest distance from the vertical projection of the machine's center of gravity to any point on the boundary of the support pattern. If stable, the stability margin is positive.

4. The Working Volume

The *working volume* of a leg is a collection of points relative to the body which can be reached by the foot of that leg.

5. The Temporal Kinematic Margin

The *temporal kinematic margin* is the time remaining until a particular leg would reach the boundary of its working volume if a particular foothold were used as a support point.

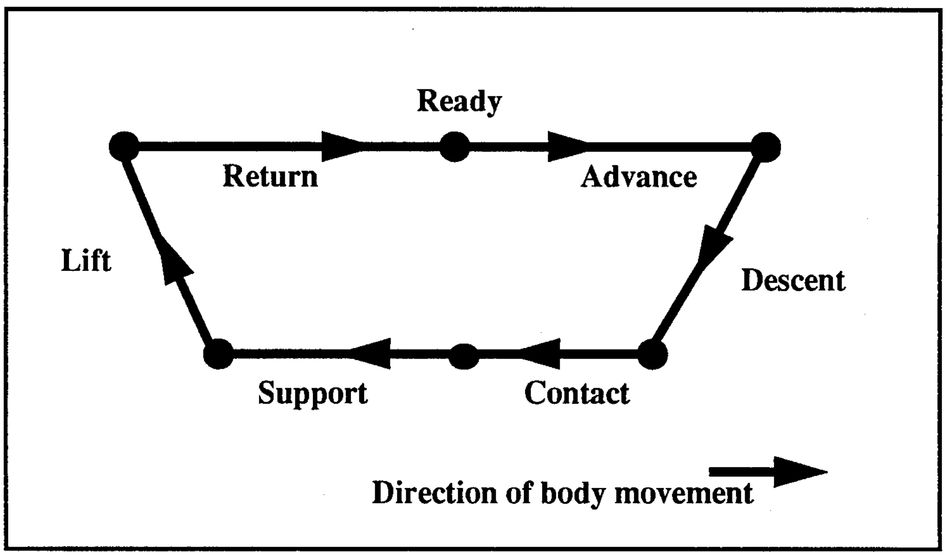


Figure 4: States of the Leg Control Machine

The static stability of a walking machine is determined by the position of the body's center of gravity relative to the support pattern. The body may move only if its stability margin and the kinematic margins of its supporting legs are positive. The mobility of the

vehicle may be improved by adjusting the support pattern or the position of the center of gravity with respect to the support pattern. Figure 5 [Ref. 1] provides an example. [Ref. 5] outlines a similar approach for the general class of hexapod walking machines.

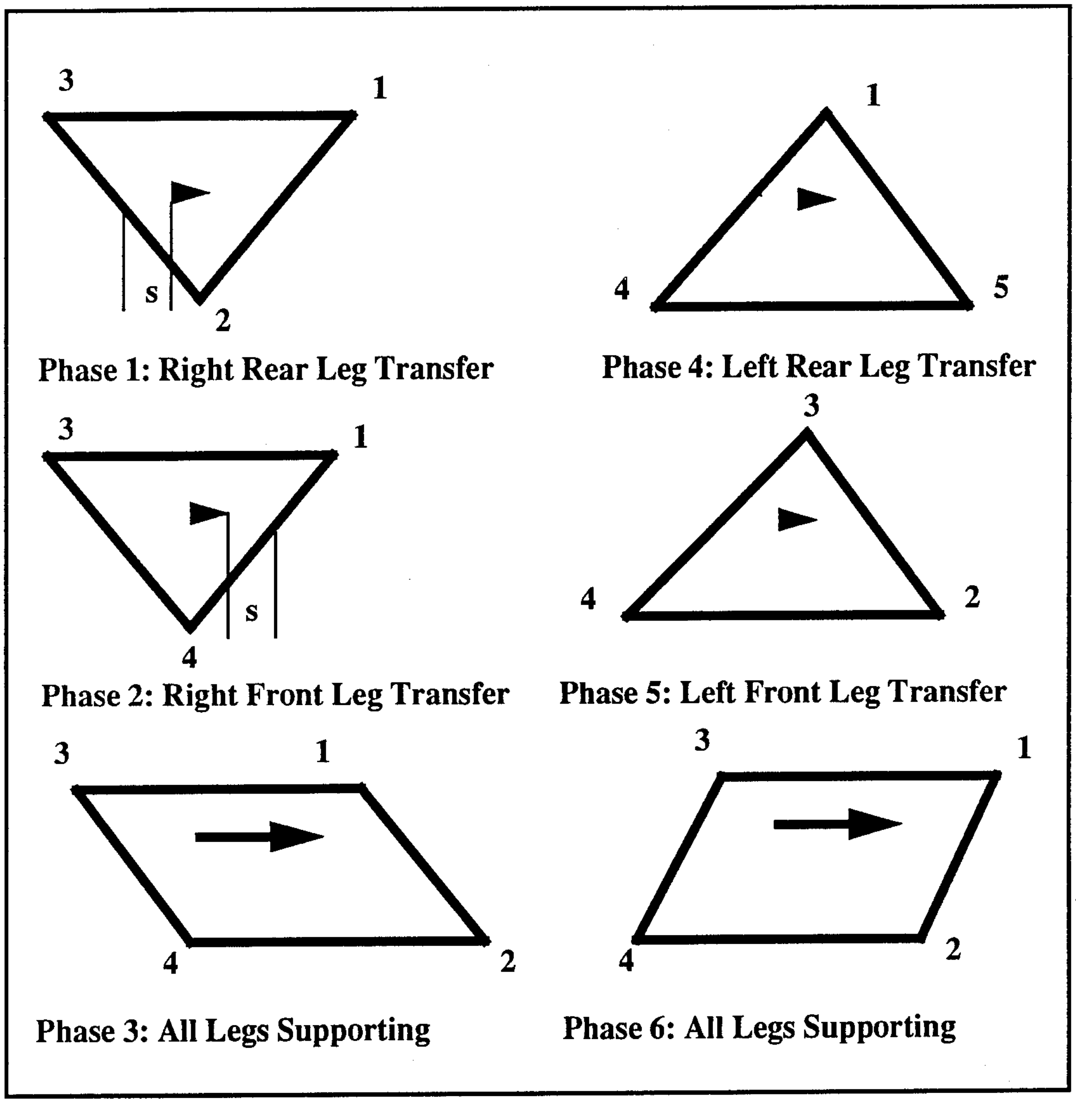


Figure 5: Support Patterns of a Quadruped Crawl Gait

[Ref. 6] also discusses free gait algorithms and defines several additional concepts:

6. Motion Trace

The *motion trace* of a vehicle is the desired trajectory for the vehicle's center of gravity.

7. Existence Segment of a Foothold

The *existence segment of a foothold* is the largest segment of the motion trace which allows a foothold to serve as a support point for a given leg of the vehicle.

8. Kinematic Margin

The *kinematic margin* for a foothold is the arc length along a motion trace from the center of gravity of the vehicle to the forward end of existence segment of the foothold.

9. Stability Segment

The *stability segment* of a support pattern is the largest segment of the motion trace such that when the vehicle's center of gravity is on this trace, the vehicle is statically stable.

10.Existence Segment of a Support Pattern

The *existence segment of a support pattern* is a maximal segment of the motion trace where:

a) the kinematic margins of all support points is greater than zero, and

b) the locomotion system is statically stable.

Movement of the vehicle along a motion trace is statically stable only if the existence segments of successive support patterns overlap. Such a support state sequence is considered to be *feasible*.

11.Optimal Support Point Sequence

An *optimal support point sequence* for motion along a motion trace is one such that:

a) the support state sequence is feasible, and

b) a specified criterion function is maximized or minimized.

C. AQUAROBOT

Research at the Naval Postgraduate School pertaining to the Aquarobot vehicle began in 1992 under a grant from the National Science foundation. Five M.S. theses and one Ph.D dissertation have focused on various aspects of this walking machine. The research conducted falls into two general categories [Ref. 7]:

1. Simulation

Simulation allows for creation of a virtual environment for testing control software before it is ever actually installed on Aquarobot. This saves production and implementation costs as well as improving design ease and efficiency. [Ref. 8] investigates the object-oriented representation of walking robot kinematics in CLOS and C++. It outlines the first stage in the development of a graphical simulation of the Aquarobot. The forward kinematics model is based on the modified Danevit-Hartenberg method [Ref. 13] of joint representation.

[Ref. 2] develops a 3-D visual simulator to render Aquarobot in real-time. This graphical model is a highly detailed and more realistic replacement for the stick-figure animation introduced in [Ref. 8]. It utilizes the Silicon Graphics toolkit, *Performer,* and tests ground detection and foot placement algorithms.

[Ref. 9] discusses the feasibility of dynamically modelling Aquarobot with respect to real-time simulation. Based upon the kinematics model implemented in [Ref. 8], it also expands this model to include inverse kinematics and physically based modelling techniques. It makes use of springs and dampers in lieu of joint actuators for the initial study, then develops a joint actuator simulation model intended to eventually replace the springs and dampers.

[Ref. 10] takes the physical dynamics simulation model further. It introduces a complete and unsimplified representation of the dynamic forces found in Aquarobot's natural operating environment, including hydrodynamic forces. Though unlikely to run in

real-time using typical graphics processors, it offers baseline data for comparison to the more simplified dynamics model addressed in [Ref. 9].

2. Control Software

[Ref. 11] presents an algorithm for an alternating tripod gait with path planning for smooth leg motion while walking on flat terrain. Three legs are moved together and the body moves only when all feet are on the ground. [Ref. 12] develops a wave gait algorithm which allows control of individual legs while maintaining the fluid trajectory of the body. This approach makes optimal use of the range of motion for the legs while ensuring a sufficient margin of stability.

D. SUMMARY

To date, walking robots have had limited practical application even though their development has progressed to the point where the most advanced models enjoy 18 independently controlled degrees of freedom. Anticipated improvements in microcomputer coordination of joint motions to increase agility and in mechanical power distribution to lower fuel consumption are expected to have a positive effect on future innovation [Ref. 1]. However, the complexity of the leg coordination and control problem, the need to further develop and understand appropriate gait algorithms, as well as cost considerations, continue to hamper realistic implementation of these machines [Ref. 7].

The remainder of this thesis will address the concepts of joint space motion margins and stability margins as they relate to motion control and Aquarobot.

III. JOINT SPACE MOTION MARGIN

The focus of this chapter and the next is to further investigate stability and joint space motion margins as they apply to Aquarobot in order to provide a basis for future investigation of control systems and gait algorithms pertaining to hexapod walking robots. Experiments are conducted in Franz Common Lisp using Allegro Common Windows for graphical output. Lisp provides an effective, easily understood method by which to prototype and implement the kinematic model and control systems associated with Aquarobot.

A. PHYSICAL DESCRIPTION

Aquarobot is an underwater walking robot, also known as a Remotely Operated Vehicle (ROV), equipped with six "insect-type" legs and a video camera arm. Each leg and the camera arm have three rotary joints comprising a total of three degrees of freedom per limb. Each joint is moved by computer input to a DC motor which controls a reduction gear attached to the joint. Disc-shaped footpads are attached to each leg by a ball joint. These footpads are not manipulated directly, but instead are oriented by the effects of gravity, the nature of the terrain encountered, and hydrodynamic forces. The robot is intended to move according to a path determined by navigational beacons utilizing a stepping sequence calculated by a control station located at the surface. Pressure sensitive sensors located on the lower limb segments gather depth data. Information is passed back and forth via a tether connecting the control station to Aquarobot. [Ref. 8]

Aquarobot currently operates under the control of a wave gait algorithm which allows manipulation of individual legs while maintaining a fluid trajectory of the body. Additionally, an alternating tripod gait has been developed with path planning for smooth leg motion while walking on flat terrain. Both of these control schemes are *synchronous*, i. e., lifting and placing of feet occurs according to a fixed time schedule, independent of body velocity. [Ref. 9]

B. THE KINEMATICS MODEL

1. Kinematics

Kinematics is "the science of motion which treats motion without regard to the forces which cause it" [Ref. 8].

2. Forward Kinematics

Forward kinematics calculates the position and orientation of the links of a limb using azimuth, elevation, and roll angles and link lengths [Ref. 13]. The position of the *end-effector*, or free end of a limb, may be calculated based upon the joint orientation angles given as parameters.

The Lisp representation of Aquarobot is defined and maintained within an object-oriented hierarchy where a user defined type ***rigid-body*** sits at the top. ***Aquarobot,*** itself, is also defined as a top level class and is made up of the parts ***aquarobot-body*** and ***links*** which are all rigid-bodies. Additionally, a class ***strobe-camera*** of type rigid-body has been developed to generate a graphical simulation of the Aquarobot model. The strobe-camera body coordinate system is such that the positive Z-axis is out the bottom of the camera, the positive Y-axis is to the right, and the positive X-axis points along the camera optical axis. In contrast, the Common Windows graphics window used to display camera images adopts the usual graphics convention with the positive Y-axis up, the X-axis to the right, and the Z-axis pointing out of the screen.

Much of the forward kinematics model used in this thesis is based upon the model developed in [Ref. 8]. Each of Aquarobot's six legs consist of a set of four connected links. The first link, link 0, is an imaginary link that extends from the center of the body to the point where the leg attaches to the body. Link 1, link 2, and link 3 make up the actual physical representation of the leg.

The Danevit-Hartenberg method is used to calculate the cumulative effect of joint motion on the set of links constituting the leg. Under this method, a transformation-matrix is used to delineate the position of the *inboard joint* of a link relative to the *outboard joint*

of the link. The inboard joint associated with a link is that joint closest to a specified reference body. The outboard joint is the joint closest to the end effector. The coordinate origin for a link is at the outboard joint and is aligned so that a joint rotation is equivalent to a z-axis rotation. The transformation matrix is called an *A-matrix* and its parameters are link length, link twist, inboard joint offset, and inboard joint angle as inputs. To transform a joint, all preceding and current link transformation matrices must be multiplied together. [Ref. 8]

Homogeneous, or *H-matrix*, transformations redefine body coordinates to world coordinates and are applied top-down within the rigid-body component hierarchy. When Aquarobot's body moves, its coordinate system is redefined to the world coordinate system and the effect of the body's motion is propagated down the links of the legs via the respective H-matrices. In this manner, kinematics allows foot position to be calculated using successive matrix transformations. [Ref. 8]

Figures 6 and 7 show the initial configuration of Aquarobot as simulated in this project. Solid circles indicate supporting legs.

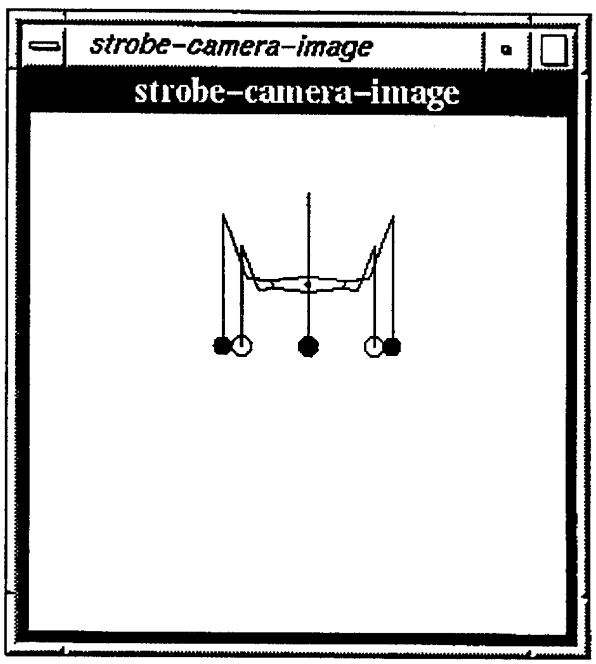


Figure 6: Aquarobot's Initial Simulation Configuration (Side View)

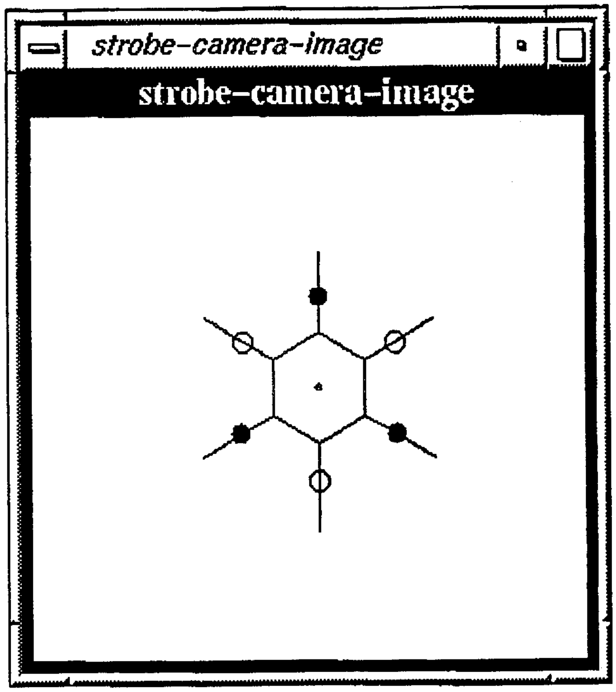


Figure 7: Aquarobot's Initial Simulation Configuration (God's Eye).

3. Inverse Kinematics

Inverse kinematics calculates the joint orientation angles when position and orientation are known [Ref. 8]. If the position of inboard end of the limb is known, and the position of the end-effector is known, then the limb joint angles can be computed.

The inverse kinematics utilized in this thesis are outlined in [Ref. 9]. While the forward kinematics model provides a means by which to calculate foot position given body position and joint angles, the inverse kinematics model allows determination of joint angles given body and foot positions. Again, this model takes advantage of the Danevit-Hartenberg method to calculate cumulative transformations.

The joints angles to be solved for are denoted as *theta 1*, *theta 2*, and *theta 3*. Theta 1 is measured as a Z-axis rotation from the X-axis where the joint serves as the origin. This is shown in Figure 8 and the angle is calculated as [Ref. 9]:

$$\theta_1 = \operatorname{atan}\left(\frac{yfoot}{xfoot}\right) \tag{3.1}$$

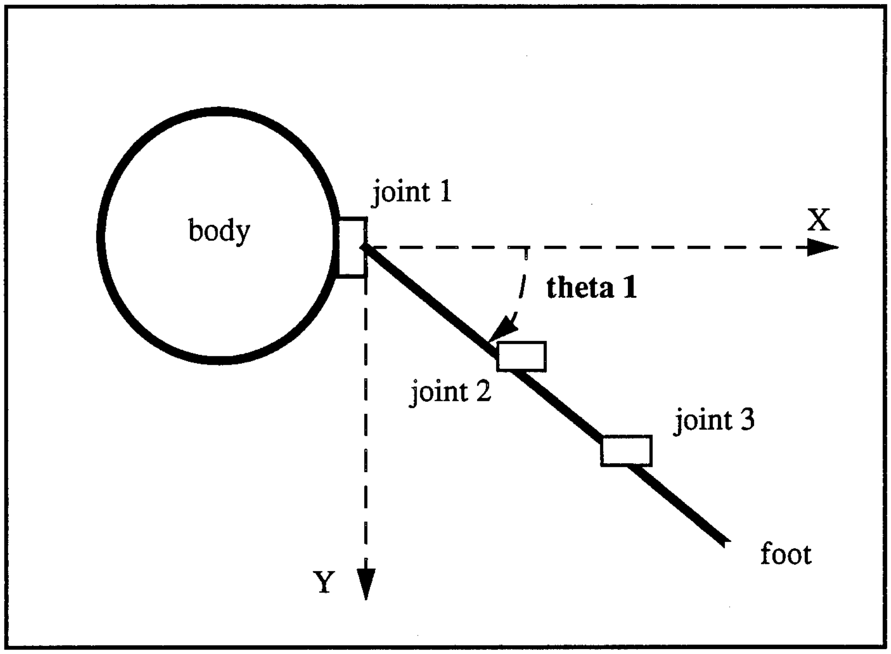


Figure 8: Inverse Kinematic Calculations - Theta 1

To compute theta 2 and theta 3, joint 2 serves as the coordinate origin and, from Figure 9 [Ref. 9], apply the Law of Cosines

$$a^2 = b^2 + c^2 + 2bc\cos\theta \tag{3.2}$$

to generate the required angles as follows [Ref. 9]:

$$\theta_2 = \operatorname{acos}\left[\frac{L_2^2 + c^2 - L_3^2}{2L_2 c}\right] - \operatorname{asin}\left[\frac{zfoot}{c}\right] \tag{3.3}$$

and,

$$\theta_3 = \operatorname{acos}\left[\frac{L_2^2 + L_3^2 - c^2}{2L_2L_3}\right] - \pi \tag{3.4}$$

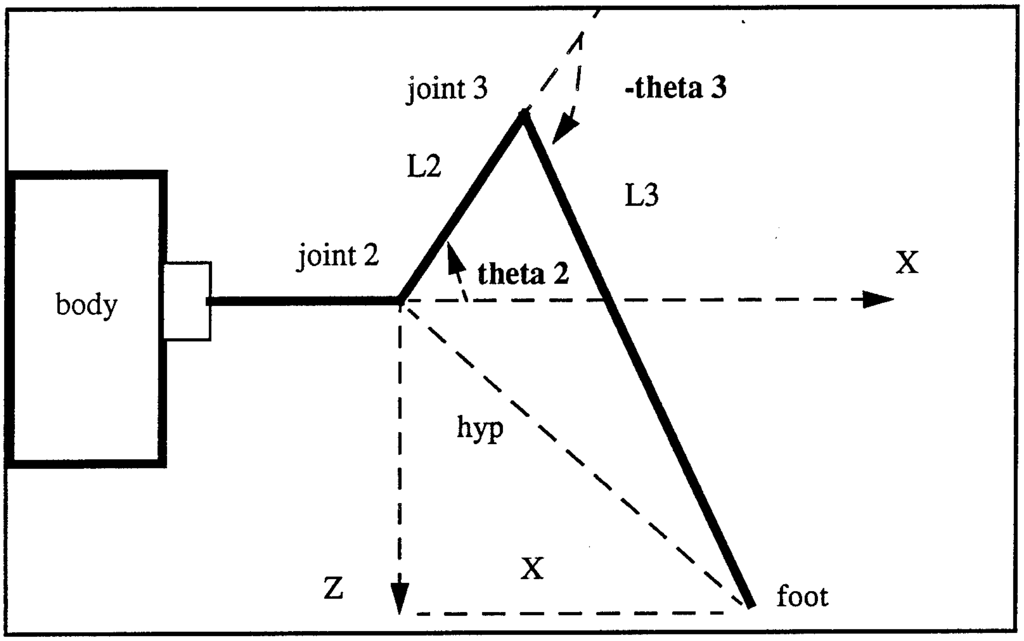


Figure 9: Inverse Kinematic Calculations - Theta 2, Theta 3

In order to ensure that no joint is rotated thorough a singular condition and that no leg is programmed to do something it is physically incapable of doing, constraints have been added to the inverse kinematics model whereby the distance between joint 2 and the foot (called hyp) cannot be greater than the sum of link length 2 (L2) and link length 3 (L3) or less than the difference between the two link lengths. If hyp exceeds these thresholds, theta 2 and theta 3 are set to either the joint minimum or maximum, whichever is applicable.

C. JOINT LIMITS AND THE JOINT SPACE MOTION MARGIN

When the body of a walking machine such as Aquarobot is moved in any way, the consequences of such a movement need to be accounted for, planned for, and acted upon with respect to the remainder of the model. The effect of the structural capabilities of the body, legs, and joints on the overall motion of the model must be considered individually and pursuant to the total model in order to accurately effect the kinematics involved. Specifically, the joint space motion margin and the stability margin must be explored.

1. Joint Limit

A *joint limit* is the upper or lower bound on the angle of rotation associated with a joint.

Each of the four links which constitute a leg in the Aquarobot kinematics model has a joint associated with it. Each of these joints has a minimum and maximum joint angle which limits the motion of the joint. This constrains movement of the leg associated with the joint as well as the body.

2. Joint Kinematic Margin

The *joint kinematic margin* is the distance in degrees until a joint reaches its minimum or maximum joint limit. Both the distance to the minimum and the maximum joint limit must be calculated for each joint of each leg. The smaller of these two distance calculations serves as the joint kinematic margin for the specified joint of the specified leg.

3. Joint Space Motion Margin

The *joint space motion margin* is the minimum joint kinematic margin among all joint kinematic margins which comprise the model. It must be taken into consideration and accounted for whenever motion is called for.

This requirement is addressed with respect to Aquarobot by the procedure **change-posture-incremental** which sits at the top of a functional and object-oriented hierarchy and works downward as follows:

a. change-posture-incremental

Change-posture-incremental is a top level function which takes as its input an aquarobot object and an increment list. The increment list consists of the translational amounts and rotation angles the body of the robot is to be moved by. The purpose of **change-posture-incremental** is to ensure both the joint space motion margin and stability margin of the model are not exceed before any type of movement by any portion of the Aquarobot is allowed. To this end, **change-posture-incremental** calls **change-posture-incremental2.**

b. change-posture-incremental2

Change-posture-incremental2 is responsible for ensuring that the model is in a state from which movement is allowed and if so, generating the requested movement. Motion is accomplished by moving the body of Aquarobot as specified in the increment list, maintaining the current foot position of those legs which belong to the current support pattern, and adjusting leg joint angles accordingly. This function calls the functions **motion-complete, re-initialize,** and **move-incremental.**

c. motion-complete

Motion-complete checks the status of all limb motion-complete-flags. If a flag is equivalent to nil, one or more links which constitute the leg have reached a joint-limit. Any further movement of that joint and the model as a whole is precluded. If all flags are true, movement from the current state is possible.

d. re-initialize/re-initialize-leg

Provided a motion is a viable one, the Aquarobot's body is moved in accordance with the increment list by the function **move-incremental.** If a leg is part of the support pattern, it is then adjusted by the function **re-initialize** which calls the lower level function **re-initialize-leg.** The leg maintains its current foot position via a call to **aqua-inv-kin.** This function calculates the new joint angles for the leg based on the current

foot position and the new body position and orientation. If a leg is not part of the support pattern, a call to **initialize-leg** moves the leg with the body in accordance with the increment-list and the leg's current configuration.

Once all joints and legs have been checked and adjusted as necessary, and the Aquarobot's position and orientation revised if appropriated, either an error message is printed if a joint kinematic margin has been reached anywhere in the model or the joint space motion margin is calculated by a call to **calc-kinematic-margin**.

e. calc-kinematic-margin

Calc-kinematic-margin evaluates all joints to determine which has the smallest distance remaining to a minimum or maximum joint limit among those legs which constitute the support pattern. It accomplishes this via calls to **calc-kinematic-margin-leg**, which evaluates the joint kinematic margins for a specific leg. **Calc-kinematic-margin-leg** calls **calc-kinematic-margin-link,** which calculates the actual joint kinematic margin for each joint.

f. print-kinematic-margin

Print-kinematic-margin outputs the joint space motion margin if the joint space motion margin has not yet been reached. If it has been reached, the functions **print-error-kmargin-joint, print-joint-limit-leg,** and **print-joint-limit-link** return the legs and joints which have reached their limits.

g. kinematic-safety-margin

The purpose of the **kinematic-safety-margin** is to provide the model with a user defined cushion or reaction time prior to reaching the actual joint space motion margin. It is set by a call to the function **specify-safety-margins.** The **kinematic-safety-margin** is in degrees. The default value is zero.

D. SUMMARY

This chapter introduces the notion of the joint space motion margin. The joint space motion margin of a walking machine utilizes the simulation model and inverse kinematics to calculate viable movements for the joints which constitute the leg model and the model for the vehicle as a whole. So long as a joint minimum or maximum has not been reached, motion is possible. The joint space motion margin keeps track of the joint distance closest to approaching a joint minimum or maximum. This ensures that the system simulates only those movements it is realistically capable of achieving, allows the model to react to an impending limit appropriately, and provides a first step toward creating useful and effective gait algorithms for the actual vehicle. The next step in simulating realistic motion is defining and implementing the concept of a stability margin for the vehicle. This is discussed in the next chapter.

Lisp code which generates the joint space motion margin and simulation is found in Appendix A. A sample of the graphical simulation demonstrating joint space motion margin calculations and representations can be found in Appendix B. An excerpt from that simulation is found in the next chapter.

IV. STABILITY MARGINS

In addition to the joint space motion margin, the stability margin must also be taken into account in this simulation of kinematic movement by a walking machine. This is to prevent motion which would result in Aquarobot overturning due to static instability.

A. THE STABILITY MARGIN

The *stability margin* is the shortest distance from the center of mass of a body to an edge defined by a support pattern.

1. The Stability-Safety-Margin

The purpose of the **stability-safety-margin** is to provide the model with a user defined cushion or reaction time prior to reaching an actual zero stability margin. It is set by a call to the function **specify-safety-margins.** The **stability-safety-margin** is in centimeters. The default value is zero.

B. CALCULATING DISTANCE

The first step in calculating the stability margin is to develop a method to compute the distance from a point to a line. To begin with, the equation of a plane (a line in the two dimensional case) can be defined as

$$\vec{n} \bullet \vec{x} = d \tag{4.1}$$

where *d* is the distance of the line from the origin, *n* is the unit normal vector directed away from the origin, and *x* is any point on the line. Therefore, *d* is greater than or equal to zero.

Given Equation (4.1), and the fact that for any point x_l, on or off a line, L,

$$\vec{n} \bullet \vec{x}_1 = d_1 \tag{4.2}$$

then,

$$\Delta d = d_1 - d \tag{4.3}$$

where the change in d is the distance of x_1 from L. As shown in Figure 10, if the change in d is greater than zero, x_1 is on the opposite side of L from the origin.

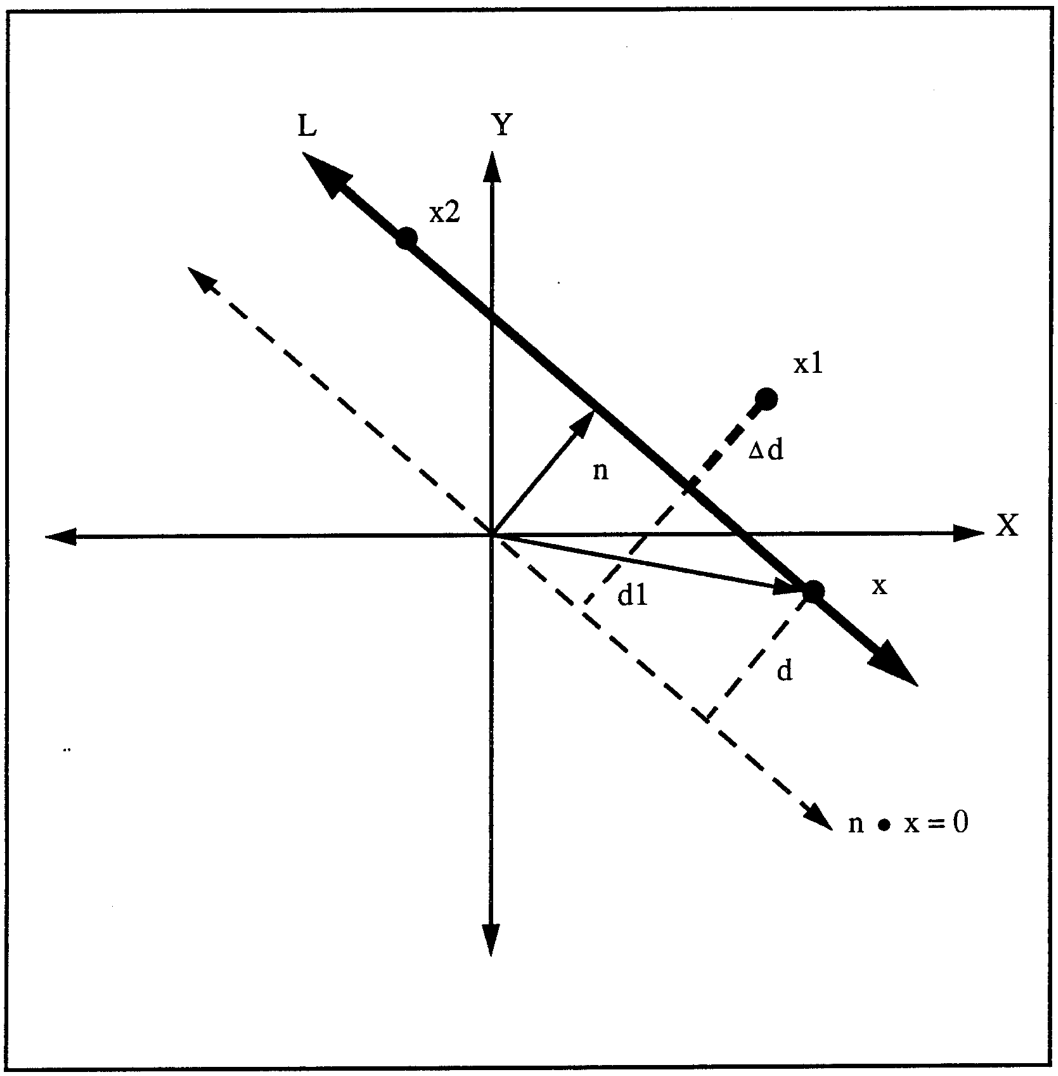


Figure 10: Stability Margin Overview

Now, given any two points defining L,

$$\vec{x}_1 \bullet \vec{n} = d \tag{4.4}$$

$$\vec{x}_2 \bullet \vec{n} = d \tag{4.5}$$

we can solve for the unknown values as follows. Subtracting Equation (4.5) from Equation (4.4) results in

$$\left(\vec{x}_1 - \vec{x}_2\right) \bullet \vec{n} = 0 \tag{4.6}$$

or

$$\Delta\vec{x} \bullet \vec{n} = 0 \tag{4.7}$$

In component form, this is

$$\Delta\vec{x} = (\Delta x \Delta y)^T \tag{4.8}$$

so,

$$(\Delta x \Delta y) \begin{pmatrix} n_x \\ n_y \end{pmatrix} = 0 \tag{4.9}$$

or

$$\Delta x n_x + \Delta y n_y = 0 \tag{4.10}$$

Thus,

$$\Delta x n_x = -\Delta y n_y \tag{4.11}$$

We now have three cases to consider with regard to the change in x and the change in y along the line, L:

<u>Case 1</u>: $$\Delta x = 0 \rightarrow n_y = 0 \rightarrow n_x = \pm 1 \tag{4.12}$$

<u>Case2</u>: $$\Delta y = 0 \rightarrow n_x = 0 \rightarrow n_y = \pm 1 \tag{4.13}$$

<u>Case3</u>: $$\Delta x \neq 0,\ \Delta y \neq 0 \rightarrow \frac{n_x}{n_y} = -\frac{\Delta y}{\Delta x} \tag{4.14}$$

In case 3, we impose the further constraint that

$$n_x^2 + n_y^2 = 1 \tag{4.15}$$

so that, using $n'_y = 1$, $n'_x = -\Delta y / \Delta x$, normalization produces the result

$$\pm n_x = \frac{-\Delta y / \Delta x}{\sqrt{1 + (\Delta y / \Delta x)^2}} \tag{4.16}$$

$$\pm n_y = \frac{1}{\sqrt{1 + (\Delta y / \Delta x)^2}} \tag{4.17}$$

The normal vector and position vector are then applied to compute distance to L as

$$d = \vec{n} \bullet \vec{x}_1 \tag{4.18}$$

If d is less than zero, reverse the sign of both components of n and recompute. This resolves the sign ambiguity. Finally, having solved for d and n, apply Equation (4.3) to compute the distance of any point from any line. This, then, is applied to the kinematic model to solve for stability margins. If d is equal to zero, the normal is confined to a right half plane so that x component values are positive. Correspondingly, positive values for n_y and n_x are chosen for horizontal and vertical lines respectively.

C. IMPLEMENTING STABILITY

1. Applying Distance Calculations to the Kinematic Model

Application of the distance calculations to Aquarobot is accomplished by dividing the legs into two sets of tripods: tripod 1 and tripod 2. Tripod 1 consists of leg 1, leg 3, and leg 5. Tripod 2 consists of leg 2, leg 4, and leg 6. To begin with, lines are calculated between adjacent feet in the tripod designated the *supporting-tripod* or support pattern as illustrated for tripod 1 in Figure 11.

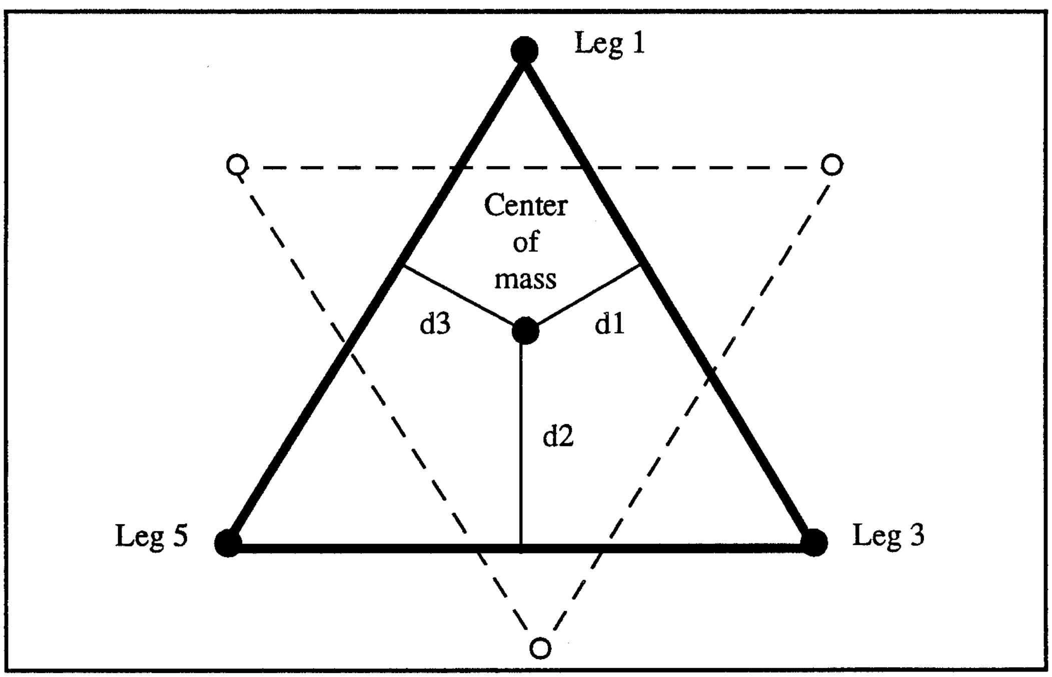


Figure 11: Distance Calculations for Aquarobot - Tripod 1

The goal is to ensure that Aquarobot's center of mass, or *location,* remains inside the triangle generated by the supporting-tripod. In other words, given the *current-foot-position* of the legs which constitute the supporting-tripod and the position of the center of mass, it is necessary to establish if the center of mass lies inside the support pattern. This can be determined by comparing the sign of d for the center of mass to that of the vertex which is off the line. This is done for all three vertices and all three lines which constitute the support pattern. If the signs are the same, the center of mass is inside the supporting-tripod. If the signs are different, the center of mass lies outside the supporting-tripod.

The distance calculations are implemented within the kinematic model and simulated as follows:

a. calc-tripod1-normals/calc-tripod2-normals

Calc-tripod1-normals and **calc-tripod2-normals** return the normals of the current supporting-tripod - either tripod 1 or tripod 2. The normals are calculated for the line formed by adjacent legs based on the legs' current foot position in accordance with the calculations outlined in Equations (4.12) through (4.17) above. Normals are returned for all three line segments which constitute the tripod.

b. calc-tripod1-distances/calc-tripod2-distances

Calc-tripod1-distances and **Calc-tripod2-distances** takes the normals returned by **calc-tripod1-normals/calc-tripod2-normals** and computes the distance from the origin to the lead leg of each tripod line segment which constitutes the supporting-tripod. **Calc-tripod1-distances/calc-tirpod2-distance** obtain the distance to each tripod line segment in the supporting-tripod as outline in Equation (4.18).

c. calc-body-distances

Calc-body-distances computes the distance of Aquarobot's body's center of mass from the origin utilizing the Aquarobot's location, or center of mass position, and

the normals found for each tripod line segment in the supporting-tripod as outline in Equation (4.18).

2. Implementing the Stability Margin

Once distance calculation considerations have been addressed, the stability margin of the model may be calculated. This is accomplished by **stability-margin** as follows:

a. stability-margin

Stability-margin sets Aquarobot's **instability-flag** based on the results of a call to **calc-stability-margin**. It is set to true if a stability margin has been reached or exceeded and false if the stability margin has not been reached. A stability margin has been exceeded if the value computed by **calc-stability-margin** is less than zero.

b. calc-stability-margin

Calc-stability-margin determines the smallest distance from the center of mass of the body to a line segment in the support pattern by finding the difference between body distance returned by **calc-body-distances** and the tripod line segment distances returned by **calc-tripod1-distances/calc-tripod2-distances** as outlined in Equation (4.3).

c. print-stability-margin-segment

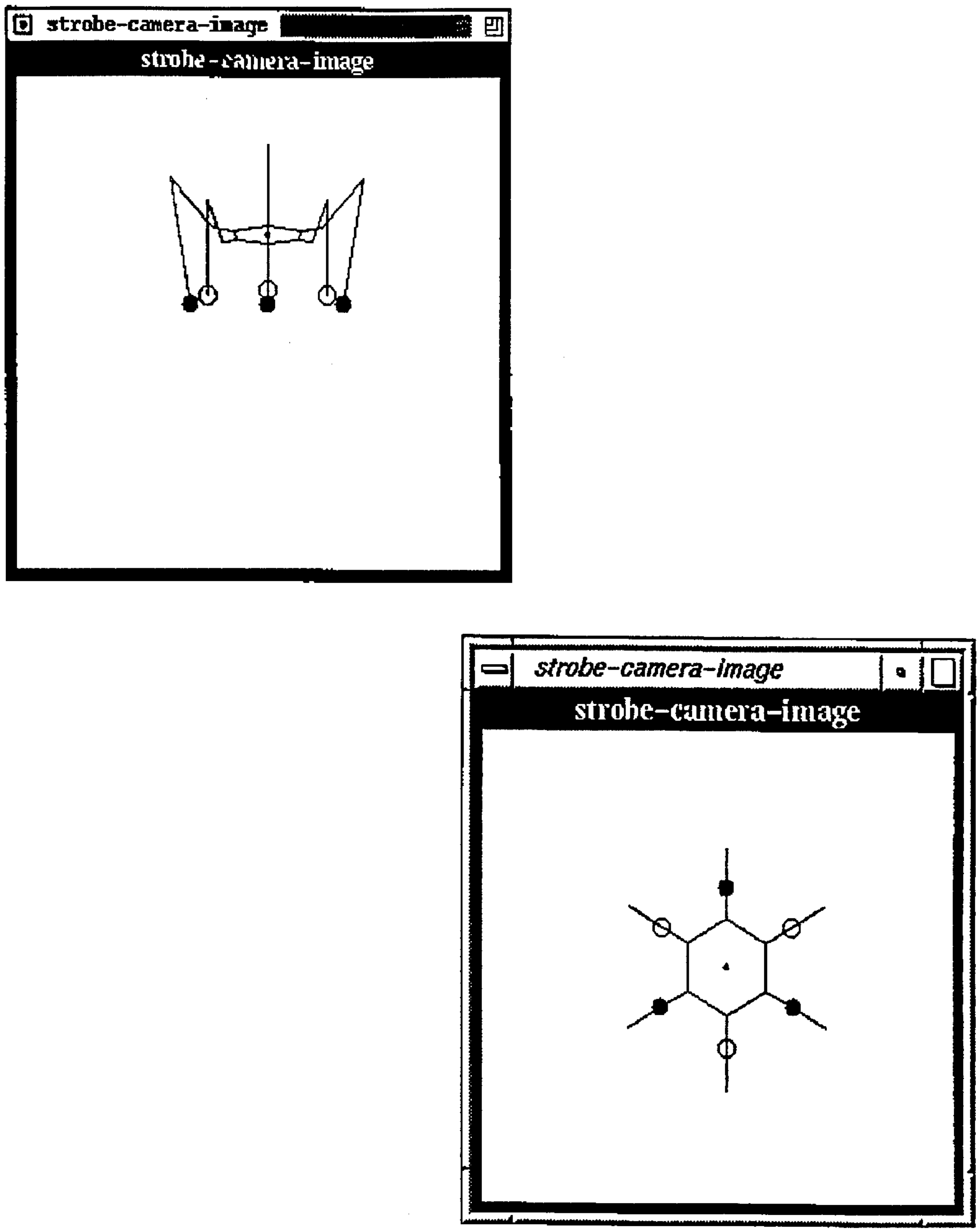
Print-stability-margin-segment prints the stability margin if the Aquarobot is in a stable state or returns the side of instability if it is in an unstable state via calls to **side-of-instabillity1, side-of-instabillity2, list-stability-margins1, list-stability-margins2**, and **calc-stability-margin.**

D. SUMMARY

This chapter introduces the notion of a walking machine's stability margin. It develops a means to calculate the distance of the body's center of mass to each edge of a tripod support pattern, to ascertain whether a requested movement maintains the overall system stability of the vehicle. So long as the center of mass lies inside the triangular

support pattern, the vehicle is stable and motion is possible. The stability margin keeps track of the distance to the closest edge constituting the tripod support pattern. This ensures that the system simulates only realistic movement, allows the model to react to impending instability, and is necessary to creating useful and effective gait algorithms for the actual vehicle.

The Lisp code necessary to implement the stability margin calculations outlined here can be found in Appendix A. Figure 12 shows an excerpt of a simulation of the stability margin calculations implemented in this thesis. The entire sample simulation of stability and joint space motion margins is found in Appendix B.



Mouse Event: SHFT+Right-click: translate-z
stability margin] = 13.4606 degrees
Stability margin = 38.7587 cm

Figure 12: Example: Joint Space Motion and Stability Margins

V. SUMMARY AND CONCLUSIONS

A. SUMMARY

This thesis has explored the concepts of the joint space motion margin and the stability margin with respect to hexapod walking machines. It begins with a brief survey of previous work accomplished in the area of walking machines, motion control, and the Aquarobot vehicle. Ground-breaking research conducted on early walking machines such as the General Electric Quadruped Transporter, the OSU and MGU Hexapod Walking Machines, and the Adaptive Suspension Vehicle is reviewed. Previous contributions to and fundamental concepts associated with kinematics and motion control are outlined, offering prerequisite definitions and the basic ideas necessary for further investigation. Prior research pertaining to the Aquarobot is discussed in detail. This research falls into two general categories: simulation and control software. A detailed physical description of the actual Aquarobot is provided as well.

After a thorough review of the kinematics model - both forward and inverse kinematics - the notion of a joint space motion margin and the associated concepts of joint limits and joint kinematic margins are defined. In short, the joint space motion margin of a walking machine is the minimum distance in degrees until some joint hits a joint limit chosen from among all joints comprising the kinematic model. How these concepts were applied to a simulation of Aquarobot in Franz Common Lisp/Allegro Common Window is reviewed in detail.

Finally, the idea of a stability margin associated with a walking machine is introduced after a detailed discussion of distance calculations and related considerations. The manner in which these computations can be specifically applied to Aquarobot's center of mass and triangular support pattern to determine if the center of mass lies inside the support pattern and therefore stable is fully outlined. How this method can be implement in Common Lisp/Common window is reviewed as well.

B. SUGGESTIONS FOR FUTURE WORK

This thesis provides a foundation for additional work in the following areas:

1. Stability Margin Considerations for Convex Hull Support Patterns

Current stability margin calculations apply only to a triangular, tripod-type support pattern. This idea, however, can be extended to encompass more complex, convex hull support patterns [Ref. 1] consisting of a greater number of footholds. This can be accomplished by comparing theoretical projections of the center of mass of the vehicle to the edges of a support pattern and calculating the minimum distance from this point to any such edge.

2. Asynchronous Tripod Gait

The model developed here can be extended to solve for an existence segment associated with the support pattern. The existence segment can be computed by assuming the body velocity does not change and then projecting the motion of the body forward until a stability margin or joint kinematic margin is reached. This model can be utilized as part of a *super real-time simulation* control method. The information obtained from the existence segment could then be used to produce an asynchronous tripod gait in which tripods would be exchanged only when required by kinematic or stability considerations.

C. CONCLUSIONS

This thesis has defined, implemented, and simulated stability and joint space motion margins in Franz Common Lisp/Allegro Common Windows for the Aquarobot vehicle. It has provided a methodology which allows the incorporation of these concepts into a kinematic model for any hexapod walking machine. It has succeeded in laying basic groundwork for future experiments in motion control, motion planning, and efficient gait algorithms for those class of vehicles.

APPENDIX A. LISP CODE

This appendix contains the LISP code for all results pertaining to Aquarobot generated in this thesis.

A. LOAD-FILES.CL

```
;************************************************************
;Load-files.cl loads in all files associated with and necessary for the
;     kinematic simulation of Aquarobot as outlined in this thesis.
;************************************************************


(defun load-aqua ()
  (load "strobe-camera.cl")
  (load "misc.cl")
  (load "link.cl")
  (load "rigid-body.cl")
  (load "robot-kinematics.cl")
  (load "aqua.cl")
  (load "aqua-leg.cl")
  (load "aqua-link.cl")
  (load "aqua-inv-kinematics.cl"))


(defun compile-aqua ()
  (compile-file "strobe-camera.cl")
  (compile-file "misc.cl")
  (compile-file "link.cl")
  (compile-file "rigid-body.cl")
  (compile-file "robot-kinematics.cl")
  (compile-file "aqua.cl")
  (compile-file "aqua-leg.cl")
  (compile-file "aqua-link.cl")
  (compile-file "aqua-inv-kinematics.cl"))


(defun load-compiled-aqua ()
  (load "strobe-camera.fasl")
  (load "misc.fasl")
  (load "link.fasl")
  (load "rigid-body.fasl")
  (load "robot-kinematics.fasl")
  (load "aqua.fasl")
  (load "aqua-leg.fasl")
  (load "aqua-link.fasl")
  (load "aqua-inv-kinematics.fasl"))
```

B. STROBE-CAMERA.CL

```
;*******************************************************
;Strobe-camera.cl contains the code necessary to generate a graphical
;    representation of Aquarobot.
;This code was written by Prof. McGhee and modified by the author.
;    All modifications are annotated with **.
;*******************************************************


(require :xcw)
(use-package :cw)
(cw:initialize-common-windows)


(defclass strobe-camera (rigid-body)
 ((focal-length
  :accessor focal-length
  :initform 6)
 (camera-window
  :accessor camera-window
  :initform (cw:make-window-stream :borders 5
                                   :left 500
                                   :bottom 500
                                   :width 300
                                   :height 300
                                   :title "strobe-camera-image"
                                   :activate-p t))
 (H-matrix
  :initform (homogeneous-transform 0 0 0 -300 0 0));**
 (inverse-H-matrix
  :accessor inverse-H-matrix
  :initform (inverse-H (homogeneous-transform 0 0 0 -300 0 0)));**
 (enlargement-factor
  :accessor enlargement-factor
  :initform 30)))


(defun create-camera-1 ();;**
 (setf camera-1 (make-instance 'strobe-camera))
 (queue-mouse camera-1))


(defmethod queue-mouse ((camera strobe-camera));;**
 (cw:modify-window-stream-method (camera-window camera) :left-button-down
    :after 'mouse-handler)
 (cw:modify-window-stream-method (camera-window camera)
    :after 'mouse-handler) :middle-button-down
 (cw:modify-window-stream-method (camera-window camera) :right-button-down
    :after 'mouse-handler))
```

```
(defun mouse-handler (wstream cw:mouse-state &optional event);;**
  (format t "In mouse-handler button: ~a~%" (mouse-button-state))
  (cond ((eql (cw:mouse-button-state) 128)      ; Left-click
         (format t "Mouse Event: Left-click:  translate+x~%")
         (change-posture-incremental aqua-1 '(0 0 0 10 0 0)))
        ((eql (cw:mouse-button-state) 129)      ; Left-click & CNTRL key
         (format t "Mouse Event: CNTRL+Left-click: rotate+x~%")
         (change-posture-incremental aqua-1 '(0 0 .1 0 0 0)))
        ((eql (cw:mouse-button-state) 144)      ; Left-click & SHFT
         (format t "Mouse Event: SHFT+Left-click: translate-x~%")
         (change-posture-incremental aqua-1 '(0 0 0 -10 0 0)))
        ((eql (cw:mouse-button-state) 132)      ; Left-click & ALT
         (format t "Mouse Event: ALT+Left-click: rotate-x~%")
         (change-posture-incremental aqua-1 '(0 0 -.1 0 0 0)))
        ((eql (cw:mouse-button-state) 64)       ; Middle-click
         (format t "Mouse Event: Middle-click:  translate+y~%")
         (change-posture-incremental aqua-1 '(0 0 0 0 10 0)))
        ((eql (cw:mouse-button-state) 65)       ; Middle-click & CNTRL key
         (format t "Mouse Event: CNTRL+Middle-click:  rotate+y~%")
         (change-posture-incremental aqua-1 '(0 .1 0 0 0 0)))
        ((eql (cw:mouse-button-state) 80)       ; Middle-click & SHFT
         (format t "Mouse Event: SHFT+Middle-click:  translate-y~%")
         (change-posture-incremental aqua-1 '(0 0 0 0 -10 0)))
        ((eql (cw:mouse-button-state) 68)       ; Middle-click & ALT
         (format t "Mouse Event: ALT+Middle-click:  rotate-y~%")
         (change-posture-incremental aqua-1 '(0 -.1 0 0 0 0)))
        ((eql (cw:mouse-button-state) 32)       ; Right-click
         (format t "Mouse Event: Right-click:  translate+z~%")
         (change-posture-incremental aqua-1 '(0 0 0 0 0 10)))
        ((eql (cw:mouse-button-state) 33)       ; Right-click & CNTRL key
         (format t "Mouse Event: CNTRL+Right-click:  rotate+z~%")
         (change-posture-incremental aqua-1 '(.1 0 0 0 0 0)))
        ((eql (cw:mouse-button-state) 48)       ; Right-click & SHFT
         (format t "Mouse Event: SHFT+Right-click:  translate-z~%")
         (change-posture-incremental aqua-1 '(0 0 0 0 0 -10)))
        ((eql (cw:mouse-button-state) 36)       ; Right-click & ALT
         (format t "Mouse Event: ALT+Right-click:  rotate-z~%")
         (change-posture-incremental aqua-1 '(-.1 0 0 0 0 0)))
        (t (format t "Invalid Mouse Event~%"))))


(defmethod move-object ((camera strobe-camera) azimuth elevation roll x y z)
 (setf (H-matrix camera) (homogeneous-transform azimuth elevation roll x y z))
 (setf (inverse-H-matrix camera) (inverse-H (H-matrix camera))))


(defmethod take-picture ((camera strobe-camera) (body rigid-body))
 (let ((camera-space-node-list (mapcar #'(lambda (node-location)
       (post-multiply (inverse-H-matrix camera) node-location))
              (transformed-node-list body))))
   (dolist (polygon (polygon-list body))
    (clip-and-draw-polygon camera polygon camera-space-node-list))))


(defmethod erase-window ((camera strobe-camera))
  (cw:clear (camera-window camera)))
```

```
(defmethod clip-and-draw-polygon
 ((camera strobe-camera) polygon node-coord-list)
 (do* ((initial-point (nth (first polygon) node-coord-list))
      (from-point initial-point to-point)
      (remaining-nodes (rest polygon) (rest remaining-nodes))
      (to-point (nth (first remaining-nodes) node-coord-list)
           (if (not (null (first remaining-nodes)))
              (nth (first remaining-nodes) node-coord-list))))
     ((null to-point)
          (draw-clipped-projection camera from-point initial-point))
     (draw-clipped-projection camera from-point to-point)))


(defmethod draw-clipped-projection ((camera strobe-camera) from-point to-point)
 (cond ((and (<= (first from-point) (focal-length camera))
           (<= (first to-point) (focal-length camera))) nil)
      ((<= (first from-point) (focal-length camera))
       (draw-line-in-camera-window camera
        (perspective-transform camera
          (from-clip camera from-point to-point))
        (perspective-transform camera to-point)))
      ((<= (first to-point) (focal-length camera))
       (draw-line-in-camera-window camera
        (perspective-transform camera from-point)
        (perspective-transform camera
          (to-clip camera from-point to-point))))
      (t (draw-line-in-camera-window camera
        (perspective-transform camera from-point)
        (perspective-transform camera to-point)))))


(defmethod from-clip ((camera strobe-camera) from-point to-point)
 (let ((scale-factor (/ (- (focal-length camera) (first from-point))
                 (- (first to-point) (first from-point)))))
     (list (+ (first from-point)
          (* scale-factor (- (first to-point) (first from-point))))
        (+ (second from-point)
          (* scale-factor (- (second to-point) (second from-point))))
        (+ (third from-point)
          (* scale-factor (- (third to-point) (third from-point)))) 1)))


(defmethod to-clip ((camera strobe-camera) from-point to-point)
 (from-clip camera to-point from-point))


(defmethod draw-line-in-camera-window ((camera strobe-camera) start end)
 (cw:draw-line (camera-window camera)
         (cw:make-position :x (first start) :y (second start))
         (cw:make-position :x (first end) :y (second end))
         :brush-width 0))
```

```
(defmethod perspective-transform ((camera strobe-camera) point-in-camera-space)
 (let* ((enlargement-factor (enlargement-factor camera))
     (focal-length (focal-length camera))
     (x (first point-in-camera-space)) ;x axis is along optical axis
     (y (second point-in-camera-space)) ;y is out right side of camera
     (z (third point-in-camera-space))) ;z is out bottom of camera
    (list (+ (round (* enlargement-factor (/ (* focal-length y) x)))
         150)          ;to right in camera window
       (+ 150 (round (* enlargement-factor (/ (* focal-length (- z)) x))
                 ))))) ;up in camera window


(defun kill () ;;**
 (cw:kill-common-windows))


(defun reset-windows ();;**
 (kill)
 (cw:initialize-common-windows))


(defun print-intro ();;**
 (format t "
*****************************************************************
    --In order to move aquarobot, use the following key/mouse
     combinations:

        Translation                    Rotation

      left-click      =     +x   =  CNTRL+left-click
      middle-click    =     +y   =  CNTRL+middle-click
      right-click     =     +z   =  CNTRL+right-click

      SHFT+left-click  =     -x
      SHFT+middle-click =    -y
      SHFT+right-click =     -z

     Alternatively, movement may be specified as:

      (change-posture-incremental aqua increment-list)

     The only allowable first move is in the -z direction.  All
     other movements will result in an error message.

    --To specity kinematic and stability safety margins, enter
     enter the following:

      (specify-safety-margins aqua k s)

     k is in degrees and s is in cm. The default values are 0.

*****************************************************************"))


(defmethod draw-node ((camera strobe-camera) location) ;;**
  (cw:draw-circle-xy (camera-window camera) (first location)
             (second location) 5))
```

```
(defmethod draw-filled-node ((camera strobe-camera) location);;**
 (cw:draw-filled-circle-xy (camera-window camera) (first location)
                (second location) 5))


(defmethod draw-filled-node2 ((camera strobe-camera) location);;**
 (cw:draw-filled-circle-xy (camera-window camera) (first location)
                (second location) 2))


(defmethod take-picture-supporting-tripod ((camera strobe-camera) tripod);;**
  (setf f1 (world-to-body camera (first tripod)))
  (setf f2 (world-to-body camera (second tripod)))
  (setf f3 (world-to-body camera (third tripod)))
  (draw-filled-node camera (perspective-transform camera f1))
  (draw-filled-node camera (perspective-transform camera f2))
  (draw-filled-node camera (perspective-transform camera f3)))


(defmethod take-picture-nonsupporting-tripod ((camera strobe-camera) tripod);;**
  (setf f1 (world-to-body camera (first tripod)))
  (setf f2 (world-to-body camera (second tripod)))
  (setf f3 (world-to-body camera (third tripod)))
  (draw-node camera (perspective-transform camera f1))
  (draw-node camera (perspective-transform camera f2))
  (draw-node camera (perspective-transform camera f3)))


(defmethod take-picture-tripods ((camera strobe-camera) st nt);;**
  (take-picture-supporting-tripod camera st)
  (take-picture-nonsupporting-tripod camera nt))


(defmethod take-picture-center ((camera strobe-camera) center);;**
  (draw-filled-node2 camera (perspective-transform camera
                    (world-to-body camera center))))
```

C. MISC.CL

```
;*********************************************************
;Misc.cl contains miscellaneous functions associated with the
;    simulation of the Aquarobot kinematic model.
;This code is a modification of code implemented in [Ref. 9].
;    All modifications are annotated with **.
;*********************************************************


(defun sqr (x) (* x x))


(defconstant rad-to-deg-multiplier (/ 180 pi))
(defun rad-to-deg (rad) (* rad rad-to-deg-multiplier))


(defconstant deg-to-rad-multiplier (/ pi 180))
(defun deg-to-rad (deg) (* deg deg-to-rad-multiplier))


;Returns first n elements of list.
(defun ncar (n list)
  (cond ((zerop n) nil)
      (t (cons (car list) (ncar (1- n) (cdr list))))))


(defun calc-distance (n x);;**
  (setf distance (dot-product n x))
  (cond ((>= distance 0)
       (dot-product n x))
  (t (dot-product (list (- (first n)) (- (second n)) 0) x))))


(defun calc-distance2 (n x);;**
  (dot-product n x))


(defun calc-delta-xy (x1 x2);;**
  (vector-subtract x1 x2))


(defun calc-normal (x1 x2);;**
  (setf delta-xy (calc-delta-xy x1 x2))
  (cond ((zerop (first delta-xy))
       (list 1 0 0))
  (t (cond ((zerop (second delta-xy))
            (list 0 1 0))
     (t (list (calc-xnormal delta-xy)
           (calc-ynormal delta-xy)
           0))))))


(defun calc-xnormal (delta-xy);;**
  (/ (/ (- (second delta-xy)) (first delta-xy))
    (sqrt (+ 1 (sqr (/ (second delta-xy) (first delta-xy)))))))


(defun calc-ynormal (delta-xy);;**
  (/ 1 (sqrt (+ 1 (sqr (/ (second delta-xy) (first delta-xy)))))))
```

D. LINK.CL

```
;*********************************************************
;Link.cl contains the code necessary to define a link, rotary link, and
;     sliding link associated with the Aquarobot kinematic model.
;This code was written by Prof. McGhee.
;*********************************************************


(defclass link (rigid-body)
 ((motion-limit-flag
   :initform nil
   :accessor motion-limit-flag)
  (twist-angle
   :initarg :twist-angle
   :accessor twist-angle)
  (link-length
   :initarg :link-length
   :accessor link-length)
  (inboard-joint-angle
   :initarg :inboard-joint-angle
   :accessor inboard-joint-angle)
  (inboard-joint-displacement
   :initarg :inboard-joint-displacement
   :accessor inboard-joint-displacement)
  (inboard-link
   :initarg :inboard-link
   :accessor inboard-link)
  (A-matrix
   :accessor A-matrix)))


(defclass rotary-link (link)
 ((min-joint-angle
   :initarg :min-joint-angle
   :accessor min-joint-angle)
  (max-joint-angle
   :initarg :max-joint-angle
   :accessor max-joint-angle)))


(defclass sliding-link (link)
 ((min-joint-displacement
   :initarg :min-joint-displacement
   :accessor min-joint-displacement)
  (max-joint-displacement
   :initarg :max-joint-displacement
   :accessor max-joint-displacement)))
```

E. RIGID-BODY.CL

```
;*****************************************************
;Rigid-body.cl defines the class rigid-body associated with the
;    Aquarobot kinematic model.
;This code was written by Prof. McGhee and modified by [Ref. 9].
;    All modifications are annotated with **.
;*****************************************************


(defclass rigid-body
  ()
  ((location          ;The three-vector (x y z) in world coordinates.
    :initarg :location
    :accessor location)
   (velocity          ;The six-vector (u v w p q r) in body coordinates.
    :initform '(0 0 0 0 0 0);**
    :initarg :velocity
    :accessor velocity)
   (velocity-growth-rate ;The vector (u-dot v-dot w-dot p-dot q-dot r-dot).
    :accessor velocity-growth-rate)
   (forces-and-torques   ;The vector (Fx Fy Fz L M N) in body coordinates.
    :initform '(0 0 0 0 0 0)
    :accessor forces-and-torques)
   (moments-of-inertia   ;The vector (Ix Iy Iz) in principal axis coordinates.
    :initform '(1 1 1)
    :initarg :moments-of-inertia
    :accessor moments-of-inertia)
   (mass
    :initform 1
    :initarg :mass
    :accessor mass)
   (node-list   ;(x y z 1) in body coord for each node. Starts with (0 0 0 1).
    :initform '((0 0 0 1) (4 0 0 1) (2 0 0 1) (-4 0 0 1) (-5 0 -2 1)
           (-6 -1.5 -2 1) (-6 1.5 -2 1) (-2 6 -2 1) (-2 -6 -2 1))
    :initarg :node-list
    :accessor node-list)
   (polygon-list
    :initform '((1 2 3 4 5) (4 6) (7 2 8))
    :initarg :polygon-list
    :accessor polygon-list)
   (transformed-node-list ;(x y z 1) in earth coord for each node in node-list.
    :accessor transformed-node-list)
   (H-matrix
    :initform (unit-matrix 4)
    :accessor H-matrix)
   (current-time
    :accessor current-time)))


(defmethod move-object ((body rigid-body) azimuth elevation roll x y z)
  (setf (H-matrix body)
    (homogeneous-transform azimuth elevation roll x y z))
  (transform-node-list body)
  (update-position body))
```

```
(defmethod move-incremental ((body rigid-body) increment-list)
 (setf (H-matrix body)
  (matrix-multiply (H-matrix body)
          (homogeneous-transform
           (first increment-list)      ;body z rotation
           (second increment-list)    ;body y rotation
           (third increment-list)     ;body x rotation
           (fourth increment-list)    ;body x translation
           (fifth increment-list)     ;body y translation
           (sixth increment-list))))  ;body z translation
 (transform-node-list body)
 (update-position body))


(defmethod get-delta-t ((body rigid-body))
 (let* ((new-time (get-internal-real-time))
     (delta-t (/ (- new-time (current-time body)) 1000)))
  (setf (current-time body) new-time)
  delta-t))


(defmethod start-timer ((body rigid-body))
 (setf (current-time body) (get-internal-real-time)))


(defmethod update-rigid-body ((body rigid-body))     ;Euler integration.
 (let ((delta-t (get-delta-t body)))
    (update-posture body delta-t)
    (update-velocity body delta-t)
    (update-velocity-growth-rate body)))


(defmethod update-velocity-growth-rate ((body rigid-body))
 (setf (velocity-growth-rate body) ;Assumes principal axis coordinates with
  (multiple-value-bind          ;origin at center of gravity of body.
   (Fx Fy Fz L M N  u v w p q r  Ix Iy Iz) ;Declares local variables.
   (values-list                   ;Values assigned.
    (append
     (forces-and-torques body) (velocity body) (moments-of-inertia body)))
   (list (+ (* v r) (* -1 w q) (/ Fx (mass body))
        (* *gravity* (first (third (H-matrix body)))))
      (+ (* w p) (* -1 u r) (/ Fy (mass body))
        (* *gravity* (second (third (H-matrix body)))))
      (+ (* u q) (* -1 v p) (/ Fz (mass body))
        (* *gravity* (third (third (H-matrix body)))))
      (/ (+ (* (- Iy Iz) q r) L) Ix)
      (/ (+ (* (- Iz Ix) r p) M) Iy)
      (/ (+ (* (- Ix Iy) p q) N) Iz))))) ;Value returned.

(defmethod update-velocity ((body rigid-body) delta-t)
 (setf (velocity body)
  (vector-add (velocity body)
        (scalar-multiply delta-t (velocity-growth-rate body)))))
```

```
(defmethod update-posture ((body rigid-body) delta-t) ;Euler integration.
 (move-incremental body
  (list (* delta-t (sixth (velocity body)))
       (* delta-t (fifth (velocity body)))
       (* delta-t (fourth (velocity body)))
       (* delta-t (first (velocity body)))
       (* delta-t (second (velocity body)))
       (* delta-t (third (velocity body))))))


(defmethod transform-node-list ((body rigid-body))
 (setf (transformed-node-list body)
  (mapcar #'(lambda (node-location)
          (post-multiply (H-matrix body) node-location))
        (node-list body))))


(defmethod update-position ((body rigid-body))
 (setf (location body) (firstn 3 (first (transformed-node-list body)))))


(defconstant *gravity* 32.2185)


(defmethod world-to-body ((body rigid-body) xyz-pos);;**
  (ncar 3 (post-multiply (inverse-H (H-matrix body))
                (append xyz-pos '(1)))))


(defmethod body-to-world ((body rigid-body) xyz-pos);;**
  (ncar 3 (post-multiply (H-matrix body) (append xyz-pos '(1)))))
```

F. ROBOT-KINEMATICS.CL

```
;*********************************************************
;Robot-kinematics.cl defines the transformation and mathmatical
;     functions associated with the Aquarobot kinematic model.
;This code was written by Prof. McGhee.
;*********************************************************


(defun transpose (matrix)          ;A matrix is a list of row vectors.
 (cond ((null (cdr matrix)) (mapcar 'list (car matrix)))
     (t (mapcar 'cons (car matrix) (transpose (cdr matrix))))))


(defun dot-product (vector-1 vector-2)  ;A vector is a list of numerical atoms.
 (apply '+ (mapcar '* vector-1 vector-2)))


(defun vector-magnitude (vector) (sqrt (dot-product vector vector)))


(defun post-multiply (matrix vector)
 (cond ((null (rest matrix)) (list (dot-product (first matrix) vector)))
     (t (cons (dot-product (first matrix) vector)
          (post-multiply (rest matrix) vector)))))


(defun pre-multiply (vector matrix)
 (post-multiply (transpose matrix) vector))


(defun matrix-multiply (A B)     ;A and B are conformable matrices.
 (cond ((null (cdr A)) (list (pre-multiply (car A) B)))
     (t (cons (pre-multiply (car A) B) (matrix-multiply (cdr A) B)))))


(defun chain-multiply (L)       ;L is a list of names of conformable matrices.
 (cond ((null (cddr L)) (matrix-multiply (eval (car L)) (eval (cadr L))))
     (t (matrix-multiply (eval (car L)) (chain-multiply (cdr L))))))
(defun cycle-left (matrix) (mapcar 'row-cycle-left matrix))


(defun row-cycle-left (row) (append (cdr row) (list (car row))))


(defun cycle-up (matrix) (append (cdr matrix) (list (car matrix))))


(defun unit-vector (one-column length)        ;Column count starts at 1.
 (do ((n length (1- n))
     (vector nil (cons (cond ((= one-column n) 1) (t 0)) vector)))
    ((zerop n) vector)))


(defun unit-matrix (size)
 (do ((row-number size (1- row-number))
     (I nil (cons (unit-vector row-number size) I)))
    ((zerop row-number) I)))
```

```
(defun concat-matrix (A B)   ;A and B are matrices with equal number of rows.
 (cond ((null A) B)
     (t (cons (append (car A) (car B)) (concat-matrix (cdr A) (cdr B))))))


(defun augment (matrix)
 (concat-matrix matrix (unit-matrix (length matrix))))


(defun normalize-row (row) (scalar-multiply (/ 1.0 (car row)) row))


(defun scalar-multiply (scalar vector)
 (cond ((null vector) nil)
     (t (cons (* scalar (car vector))
           (scalar-multiply scalar (cdr vector))))))


(defun solve-first-column (matrix)       ;Reduces first column to (1 0 ... 0).
 (do* ((remaining-row-list matrix (rest remaining-row-list))
      (first-row (normalize-row (first matrix)))
      (answer (list first-row)
           (cons (vector-add (first remaining-row-list)
                       (scalar-multiply (- (caar remaining-row-list))
                                   first-row))
               answer)))
     ((null (rest remaining-row-list)) (reverse answer))))


(defun vector-add (vector-1 vector-2) (mapcar '+ vector-1 vector-2))


(defun vector-subtract (vector-1 vector-2) (mapcar '- vector-1 vector-2))


(defun first-square (matrix)  ;Returns leftmost square matrix from argument.
 (do ((size (length matrix))
     (remainder matrix (rest remainder))
     (answer nil (cons (firstn size (first remainder)) answer)))
    ((null remainder) (reverse answer))))
(defun firstn (n list)
 (cond ((zerop n) nil)
      (t (cons (first list) (firstn (1- n) (rest list))))))


(defun max-car-firstn (n list)
 (append (max-car-first (firstn n list)) (nthcdr n list)))


(defun matrix-inverse (M)
 (do ((M1 (max-car-first (augment M))
        (cond ((null M1) nil)
            (t (max-car-firstn n (cycle-left (cycle-up M1))))))
     (n (1- (length M)) (1- n)))
    ((or (minusp n) (null M1)) (cond ((null M1) nil) (t (first-square M1))))
    (setq M1 (cond ((zerop (caar M1)) nil) (t (solve-first-column M1))))))
```

```
(defun max-car-first (L)   ;L is a list of lists. This function finds list with
 (cond ((null (cdr L)) L) ;largest car and moves it to head of list of lists.
      (t (if (> (abs (caar L)) (abs (caar (max-car-first (cdr L))))) L
           (append (max-car-first (cdr L)) (list (car L)))))))


(defun dh-matrix (cosrotate sinrotate costwist sintwist length translate)
 (list (list cosrotate (- (* costwist sinrotate))
          (* sintwist sinrotate) (* length cosrotate))
      (list sinrotate (* costwist cosrotate)
          (- (* sintwist cosrotate)) (* length sinrotate))
      (list 0. sintwist costwist translate) (list 0. 0. 0. 1.)))


(defun homogeneous-transform (azimuth elevation roll x y z)
 (rotation-and-translation (sin azimuth) (cos azimuth) (sin elevation)
 (cos elevation) (sin roll) (cos roll) x y z))


(defun rotation-and-translation (spsi cpsi sth cth sphi cphi x y z)
 (list (list (* cpsi cth) (- (* cpsi sth sphi) (* spsi cphi))
          (+ (* cpsi sth cphi) (* spsi sphi)) x)
      (list (* spsi cth) (+ (* cpsi cphi) (* spsi sth sphi))
          (- (* spsi sth cphi) (* cpsi sphi)) y)
      (list (- sth) (* cth sphi) (* cth cphi) z)
      (list 0. 0. 0. 1.)))


(defun inverse-H (H)         ;H is a 4x4 homogeneous transformation matrix.
 (let* ((minus-P (list (- (fourth (first H)))
                  (- (fourth (second H)))
                  (- (fourth (third H)))))
       (inverse-R (transpose (first-square (reverse (rest (reverse H))))))
       (inverse-P (post-multiply inverse-R minus-P)))
      (append (concat-matrix inverse-R (transpose (list inverse-P)))
           (list (list 0 0 0 1)))))


(setf x '((1 2) (3 4)))
```

G. AQUA.CL

```
;********************************************************
;Aqua.cl defines the class aquarobot associated with the
;     Aquarobot kinematic model.
;This code was written by Prof. McGhee and modified by the author.
;     All modifications are annotated with **.
;********************************************************

(defclass aquarobot-body (rigid-body)
  ((node-list
   :initform '((0 0 0 1) (37.5 0 0 1) (18.75 32.48 0 1)
       (-18.75 32.48 0 1) (-37.5 0 0 1) (-18.75 -32.48 0 1)
           (18.75 -32.48 0 1) (37.5 0 -15 1)))
   (polygon-list
   :initform '((1 2 3 4 5 6) (1 7)))
   (H-matrix
   :initform (homogeneous-transform 0 0 0 0 0 z-init))))


(defclass aquarobot ()
 ((body
   :initform (make-instance 'aquarobot-body)
   :accessor body)
  (leg1
   :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 0))
   :accessor leg1)
  (leg2
   :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 60))
   :accessor leg2)
  (leg3
   :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 120))
   :accessor leg3)
  (leg4
   :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 180))
   :accessor leg4)
  (leg5
   :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 240))
   :accessor leg5)
  (leg6
   :initform (make-instance 'aqua-leg :leg-attachment-angle (deg-to-rad 300))
   :accessor leg6)
  (kinematic-safety-margin;;**
   :initform 0
   :accessor kinematic-safety-margin)
  (instability-flag;;**
   :initform nil
   :accessor instability-flag)
  (stability-safety-margin;;**
   :initform 0
   :accessor stability-safety-margin)
  (begin-step;;**
   :initform t
   :accessor begin-step)
  (end-step;;**
   :initform nil
   :accessor end-step)
  (supporting-tripod;;**
   :initform 1
   :accessor supporting-tripod)))
```

```
(defmethod initialize ((aqua aquarobot))
  (setf k (kinematic-safety-margin aqua))
  (transform-node-list (body aqua))
  (initialize-leg (leg1 aqua) (body aqua) k)
  (initialize-leg (leg2 aqua) (body aqua) k)
  (initialize-leg (leg3 aqua) (body aqua) k)
  (initialize-leg (leg4 aqua) (body aqua) k)
  (initialize-leg (leg5 aqua) (body aqua) k)
  (initialize-leg (leg6 aqua) (body aqua) k))

(defmethod re-initialize-all ((aqua aquarobot));**
  (setf k (kinematic-safety-margin aqua))
  (re-initialize-leg (leg1 aqua) (body aqua) k)
  (re-initialize-leg (leg2 aqua) (body aqua) k)
  (re-initialize-leg (leg3 aqua) (body aqua) k)
  (re-initialize-leg (leg4 aqua) (body aqua) k)
  (re-initialize-leg (leg5 aqua) (body aqua) k)
  (re-initialize-leg (leg6 aqua) (body aqua) k))

(defmethod re-initialize1 ((aqua aquarobot));**
  (setf k (kinematic-safety-margin aqua))
  (re-initialize-leg (leg1 aqua) (body aqua) k)
  (initialize-leg (leg2 aqua) (body aqua) k)
  (re-initialize-leg (leg3 aqua) (body aqua) k)
  (initialize-leg (leg4 aqua) (body aqua) k)
  (re-initialize-leg (leg5 aqua) (body aqua) k)
  (initialize-leg (leg6 aqua) (body aqua) k))

(defmethod re-initialize2 ((aqua aquarobot));**
  (setf k (kinematic-safety-margin aqua))
  (initialize-leg (leg1 aqua) (body aqua) k)
  (re-initialize-leg (leg2 aqua) (body aqua) k)
  (initialize-leg (leg3 aqua) (body aqua) k)
  (re-initialize-leg (leg4 aqua) (body aqua) k)
  (initialize-leg (leg5 aqua) (body aqua) k)
  (re-initialize-leg (leg6 aqua) (body aqua) k))

(defun aqua-picture ()
  (setf aqua-1 (make-instance 'aquarobot))
  (initialize aqua-1)
  (move-incremental aqua-1 null-move-list);**
  (create-camera-1)
  (take-picture camera-1 aqua-1)
  (print-intro));;**
```

```
(defmethod take-picture ((camera strobe-camera) (aqua aquarobot))
 (take-picture camera (body aqua))
 (take-picture camera (leg1 aqua))
 (take-picture camera (leg2 aqua))
 (take-picture camera (leg3 aqua))
 (take-picture camera (leg4 aqua))
 (take-picture camera (leg5 aqua))
 (take-picture camera (leg6 aqua))
 (cond ((eql (supporting-tripod aqua) 1);;**
      (take-picture-tripods camera (tripod1-position aqua) ;;**
                          (tripod2-position aqua)));;**
 (t (take-picture-tripods camera (tripod2-position aqua) ;;**
                       (tripod1-position aqua))));;**
 (take-picture-center camera (location (body aqua))));;**


(defmethod tripod1-position ((aqua aquarobot));;**
  (list (current-foot-position (leg1 aqua))
      (current-foot-position (leg3 aqua))
      (current-foot-position (leg5 aqua))))


(defmethod tripod2-position ((aqua aquarobot));;**
  (list (current-foot-position (leg2 aqua))
      (current-foot-position (leg4 aqua))
      (current-foot-position (leg6 aqua))))


(defun new-aqua-picture (azimuth elevation roll x y z) ;;**
  (move-object camera-1 azimuth elevation roll x y z)
  (erase-window camera-1)
  (new-picture))


(defun new-picture ();;**
 (erase-window camera-1)
 (take-picture camera-1 aqua-1))


(defconstant z-init -54.181866)


(defmethod move-incremental ((aqua aquarobot) increment-list)
 (setf k (kinematic-safety-margin aqua))
 (move-incremental (body aqua) (first increment-list))
 (move-incremental-leg (leg1 aqua) (second increment-list) k)
 (move-incremental-leg (leg2 aqua) (third increment-list) k)
 (move-incremental-leg (leg3 aqua) (fourth increment-list) k)
 (move-incremental-leg (leg4 aqua) (fifth increment-list) k)
 (move-incremental-leg (leg5 aqua) (sixth increment-list) k)
 (move-incremental-leg (leg6 aqua) (seventh increment-list) k))


(defconstant null-move-list '((0 0 0 0 0 0) (0 0 0) (0 0 0) (0 0 0)
 (0 0 0) (0 0 0) (0 0 0)))
```

```
(defmethod feasible-movep ((aqua aquarobot) allowable-sinkage
                          allowable-slippage)
  (and (feasible-movep (leg1 aqua) allowable-sinkage allowable-slippage)
       (feasible-movep (leg2 aqua) allowable-sinkage allowable-slippage)
       (feasible-movep (leg3 aqua) allowable-sinkage allowable-slippage)
       (feasible-movep (leg4 aqua) allowable-sinkage allowable-slippage)
       (feasible-movep (leg5 aqua) allowable-sinkage allowable-slippage)
       (feasible-movep (leg6 aqua) allowable-sinkage allowable-slippage)))


(defmethod motion-complete ((aqua aquarobot));;**
  (and (motion-complete-flag (leg1 aqua))
       (motion-complete-flag (leg2 aqua))
       (motion-complete-flag (leg3 aqua))
       (motion-complete-flag (leg4 aqua))
       (motion-complete-flag (leg5 aqua))
       (motion-complete-flag (leg6 aqua))))


(defmethod print-error-kmargin-joint ((aqua aquarobot));;**
  (cond ((eql (supporting-tripod aqua) 1)
         (cond ((not (motion-complete-flag (leg1 aqua)))
                (format t "Error: Joint limit reached in leg1: Joint~{ ~D~} ~%"
                        (print-joint-limit-link (leg1 aqua)))))
         (cond ((not (motion-complete-flag (leg3 aqua)))
                (format t "Error: Joint limit reached in leg3: Joint~{ ~D~} ~%"
                        (print-joint-limit-link (leg3 aqua)))))
         (cond ((not (motion-complete-flag (leg5 aqua)))
                (format t "Error: Joint limit reached in leg5: Joint~{ ~D~} ~%"
                        (print-joint-limit-link (leg5 aqua))))))
  (t (cond ((not (motion-complete-flag (leg2 aqua)))
            (format t "Error: Joint limit reached in leg2: Joint~{ ~D~} ~%"
                    (print-joint-limit-link (leg2 aqua)))))
     (cond ((not (motion-complete-flag (leg4 aqua)))
            (format t "Error: Joint limit reached in leg4: Joint~{ ~D~} ~%"
                    (print-joint-limit-link (leg4 aqua)))))
     (cond ((not (motion-complete-flag (leg6 aqua)))
            (format t "Error: Joint limit reached in leg6: Joint~{ ~D~} ~%"
                    (print-joint-limit-link (leg6 aqua))))))))


(defmethod calc-kinematic-margin ((aqua aquarobot));;**
  (setf k (kinematic-safety-margin aqua))
  (cond ((eql (supporting-tripod aqua) 1)
         (min (calc-kinematic-margin-leg (leg1 aqua) k)
              (calc-kinematic-margin-leg (leg3 aqua) k)
              (calc-kinematic-margin-leg (leg5 aqua) k)))
  (t (min (calc-kinematic-margin-leg (leg2 aqua) k)
          (calc-kinematic-margin-leg (leg4 aqua) k)
          (calc-kinematic-margin-leg (leg6 aqua) k)))))


(defmethod print-kinematic-margin ((aqua aquarobot));;**
  (setf margin (calc-kinematic-margin aqua))
  (cond ((motion-complete aqua)
         (format t "Kinematic margin = ~,4F degrees~%" margin))
  (t (print-error-kmargin-joint aqua))))
```

```
(defmethod change-posture-incremental2 ((aqua aquarobot) increment-list) ;;**
  (setf motion-flag (motion-complete aqua))
  (cond ((eql motion-flag t)
      (move-incremental (body aqua)
                (list (first increment-list) 0 0 0 0 0))
      (cond ((eql (supporting-tripod aqua) 1)
          (re-initialize1 aqua))
      (t (re-initialize2 aqua)))))
  (cond ((eql motion-flag t)
      (move-incremental (body aqua)
                (list 0 (second increment-list) 0 0 0 0))
      (cond ((eql (supporting-tripod aqua) 1)
          (re-initialize1 aqua))
      (t (re-initialize2 aqua)))))
  (cond ((eql motion-flag t)
      (move-incremental (body aqua)
                (list 0 0 (third increment-list) 0 0 0))
      (cond ((eql (supporting-tripod aqua) 1)
          (re-initialize1 aqua))
      (t (re-initialize2 aqua)))))
  (cond ((eql motion-flag t)
      (move-incremental (body aqua)
                (list 0 0 0 (fourth increment-list) 0 0))
      (cond ((eql (supporting-tripod aqua) 1)
          (re-initialize1 aqua))
      (t (re-initialize2 aqua)))))
  (cond ((eql motion-flag t)
      (move-incremental (body aqua)
                (list 0 0 0 0 (fifth increment-list) 0))
      (cond ((eql (supporting-tripod aqua) 1)
          (re-initialize1 aqua))
      (t (re-initialize2 aqua)))))
  (cond ((eql motion-flag t)
      (move-incremental (body aqua)
                (list 0 0 0 0 0 (sixth increment-list)))
      (cond ((eql (supporting-tripod aqua) 1)
          (re-initialize1 aqua))
      (t (re-initialize2 aqua))))))


(defmethod change-posture-incremental ((aqua aquarobot) increment-list);;**
  (change-posture-incremental2 aqua increment-list)
  (cond ((eql (supporting-tripod aqua) 1)
      (setf normals (calc-tripod1-normals aqua))
      (setf tripod-distances (calc-tripod1-distances aqua normals)))
  (t (setf normals (calc-tripod2-normals aqua))
    (setf tripod-distances (calc-tripod2-distances aqua normals))))
  (stability-margin aqua normals tripod-distances);;**
  (cond ((and (eql (motion-complete aqua) t)
          (eql (instability-flag aqua) nil))
      (new-picture)))
  (print-kinematic-margin aqua)
  (print-stability-margin-segment aqua normals tripod-distances))
```

```
(defmethod calc-tripod1-normals ((aqua aquarobot));;**
  (list (calc-normal (current-foot-position (leg1 aqua))
              (current-foot-position (leg3 aqua)))
        (calc-normal (current-foot-position (leg3 aqua))
              (current-foot-position (leg5 aqua)))
        (calc-normal (current-foot-position (leg1 aqua))
              (current-foot-position (leg5 aqua)))))

(defmethod calc-tripod2-normals ((aqua aquarobot));;**
  (list (calc-normal (current-foot-position (leg4 aqua))
              (current-foot-position (leg2 aqua)))
        (calc-normal (current-foot-position (leg6 aqua))
              (current-foot-position (leg2 aqua)))
        (calc-normal (current-foot-position (leg4 aqua))
              (current-foot-position (leg6 aqua)))))

(defmethod calc-tripod1-distances ((aqua aquarobot) normals);;**
  (list (calc-distance (first normals)
              (current-foot-position (leg1 aqua)))
        (calc-distance (second normals)
              (current-foot-position (leg3 aqua)))
        (calc-distance (third normals)
              (current-foot-position (leg5 aqua)))))

(defmethod calc-tripod2-distances ((aqua aquarobot) normals);;**
  (list (calc-distance (first normals)
              (current-foot-position (leg2 aqua)))
        (calc-distance (second normals)
              (current-foot-position (leg6 aqua)))
        (calc-distance (third normals)
              (current-foot-position (leg4 aqua)))))

(defmethod calc-body-distances ((aqua aquarobot) normals);;**
  (list (calc-distance2 (first normals)
              (location (body aqua)))
        (calc-distance2 (second normals)
              (location (body aqua)))
        (calc-distance2 (third normals)
              (location (body aqua)))))

(defmethod calc-stability-margin ((aqua aquarobot) normals tripod-distances);**
  (setf body-distances (calc-body-distances aqua normals))
  (min (- (first tripod-distances) (first body-distances))
     (- (second tripod-distances) (second body-distances))
     (+ (third tripod-distances) (third body-distances))))
```

```
(defmethod list-stability-margins1 ((aqua aquarobot));;**
  (setf normals (calc-tripod1-normals aqua))
  (setf body-distances (calc-body-distances aqua normals))
  (setf plane-distances (calc-tripod1-distances aqua normals))
  (list (- (- (first plane-distances) (first body-distances))
          (stability-safety-margin aqua))
        (- (- (second plane-distances) (second body-distances))
          (stability-safety-margin aqua))
        (- (+ (third plane-distances) (third body-distances))
          (stability-safety-margin aqua))))

(defmethod list-stability-margins2 ((aqua aquarobot));;**
  (setf normals (calc-tripod2-normals aqua))
  (setf body-distances (calc-body-distances aqua normals))
  (setf plane-distances (calc-tripod2-distances aqua normals))
  (list (- (- (first plane-distances) (first body-distances))
          (stability-safety-margin aqua))
        (- (- (second plane-distances) (second body-distances))
          (stability-safety-margin aqua))
        (- (+ (third plane-distances) (third body-distances))
          (stability-safety-margin aqua))))

(defmethod stability-margin ((aqua aquarobot) normals tripod-distances);;**
  (setf margin (calc-stability-margin aqua normals tripod-distances))
  (cond ((< margin (stability-safety-margin aqua))
        (setf (instability-flag aqua) t))))

(defmethod print-stability-margin-value ((aqua aquarobot) normals
                                        tripod-distances);;**
  (setf margin (calc-stability-margin aqua normals tripod-distances))
  (setf adj-margin (- margin (stability-safety-margin aqua)))
  (cond ((instability-flag aqua)
        (format t "Error: Stability margin reached = ~,4F cm~%" adj-margin))
  (t (format t "Stability margin = ~,4F cm~%" adj-margin))))

(defmethod print-stability-margin-segment ((aqua aquarobot) normals
                                          tripod-distances);;**
  (setf margin (calc-stability-margin aqua normals tripod-distances))
  (setf adj-margin (- margin (stability-safety-margin aqua)))
  (cond ((instability-flag aqua)
         (format t "Error: Stability margin reached: Tripod segment~{ ~D~}~%"
           (cond ((eql (supporting-tripod aqua) 1)
                  (side-of-instability1 aqua))
           (t (side-of-instability2 aqua)))))
  (t (format t "Stability margin = ~,4F cm~%" adj-margin))))
```

```
(defmethod side-of-instability1 ((aqua aquarobot));;**
  (setf margins (list-stability-margins1 aqua))
  (cond ((and (< (first margins) 0)
          (> (third margins) 0))
      (list 13))
     ((< (second margins) 0)
      (list 35))
     ((and (< (third margins) 0)
          (> (first margins) 0))
      (list 15))
     ((and (< (first margins) 0)
          (< (third margins) 0))
      (list 13 15))))


(defmethod side-of-instability2 ((aqua aquarobot));;**
  (setf margins (list-stability-margins2 aqua))
  (cond ((and (< (first margins) 0)
          (> (third margins) 0))
      (list 24))
     ((< (second margins) 0)
      (list 26))
     ((and (< (third margins) 0)
          (> (first margins) 0))
      (list 46))
     ((and (< (first margins) 0)
          (< (third margins) 0))
      (list 24 46))))


(defmethod specify-safety-margins ((aqua aquarobot) k s);;**
  (setf (kinematic-safety-margin aqua) k)
  (setf (stability-safety-margin aqua) s))
```

H. AQUA-LEG.CL

```
;*******************************************************
;Aqua-leg.cl defines the class aqua-leg associated with the
;     Aquarobot kinematic model.
;This code was written by Prof. McGhee and modified by the author.
;     All modifications are annotated with **.
;*******************************************************


(defclass aqua-leg ()
 ((leg-attachment-angle
   :initarg :leg-attachment-angle
   :accessor leg-attachment-angle)
  (link0
   :initform (make-instance 'link0)
   :accessor link0)
  (link1
   :initform (make-instance 'link1)
   :accessor link1)
  (link2
   :initform (make-instance 'link2)
   :accessor link2)
  (link3
   :initform (make-instance 'link3)
   :accessor link3)
  (motion-complete-flag
   :initform nil
   :accessor motion-complete-flag)
  (previous-foot-position
   :initform nil
   :accessor previous-foot-position)
  (current-foot-position
   :initform nil
   :accessor current-foot-position)))


(defmethod initialize-leg ((leg aqua-leg) (body aquarobot-body) k)
 (setf (inboard-link (link0 leg)) body)
 (setf (inboard-link (link1 leg)) (link0 leg))
 (setf (inboard-link (link2 leg)) (link1 leg))
 (setf (inboard-link (link3 leg)) (link2 leg))
 (rotate-link (link0 leg) (leg-attachment-angle leg) k)
 (rotate-link (link1 leg) (inboard-joint-angle (link1 leg)) k)
 (rotate-link (link2 leg) (inboard-joint-angle (link2 leg)) k)
 (rotate-link (link3 leg) (inboard-joint-angle (link3 leg)) k)
 (setf (current-foot-position leg)
      (firstn 3 (first (transformed-node-list (link3 leg))))))
```

```
(defmethod re-initialize-leg ((leg aqua-leg) (body aquarobot-body) k);;**
  (setf (inboard-link (link0 leg)) body)
  (setf (inboard-link (link1 leg)) (link0 leg))
  (setf (inboard-link (link2 leg)) (link1 leg))
  (setf (inboard-link (link3 leg)) (link2 leg))
  (rotate-link (link0 leg) (leg-attachment-angle leg) k)
  (setf new-jt-angles (aqua-inv-kin leg (current-foot-position leg)))
  (rotate-link (link1 leg) (first new-jt-angles) k)
  (rotate-link (link2 leg) (second new-jt-angles) k)
  (rotate-link (link3 leg) (third new-jt-angles) k)
  (setf (motion-complete-flag leg) (not (or (motion-limit-flag (link1 leg))
       (motion-limit-flag (link2 leg)) (motion-limit-flag (link3 leg))))))


(defmethod print-joint-limit-link ((leg aqua-leg));;**
  (setf links (list (cond ((motion-limit-flag (link1 leg))
                   1))
               (cond ((motion-limit-flag (link2 leg))
                   2))
               (cond ((motion-limit-flag (link3 leg))
                   3))))
  (cond ((and (eql (first links) 1)
          (eql (second links) nil)
          (eql (third links) nil))
       (list 1))
      ((and (eql (first links) 1)
          (eql (second links) 2)
          (eql (third links) nil))
       (list 1 2))
      ((and (eql (first links) 1)
          (eql (second links) 2)
          (eql (third links) 3))
       (list 1 2 3))
      ((and (eql (first links) nil)
          (eql (second links) 2)
          (eql (third links) nil))
       (list 2))
      ((and (eql (first links) nil)
          (eql (second links) 2)
          (eql (third links) 3))
       (list 2 3))
      ((and (eql (first links) nil)
          (eql (second links) nil)
          (eql (third links) 3))
       (list 3))
      ((and (eql (first links) 1)
          (eql (second links) nil)
          (eql (third links) 3))
       (list 1 3))))


(defmethod calc-kinematic-margin-leg ((leg aqua-leg) k);;**
  (setf new-jt-angles (aqua-inv-kin leg (current-foot-position leg)))
  (min (calc-kinematic-margin-link (link1 leg) (first new-jt-angles) k)
     (calc-kinematic-margin-link (link2 leg) (second new-jt-angles) k)
     (calc-kinematic-margin-link (link3 leg) (third new-jt-angles) k)))
```

```
(defmethod calc-kinematic-margins-leg ((leg aqua-leg) k);;**
  (setf new-jt-angles (aqua-inv-kin leg (current-foot-position leg)))
  (list (calc-kinematic-margin-link (link1 leg) (first new-jt-angles) k)
      (calc-kinematic-margin-link (link2 leg) (second new-jt-angles) k)
      (calc-kinematic-margin-link (link3 leg) (third new-jt-angles) k)))

(defmethod take-picture ((camera strobe-camera) (leg aqua-leg))
 (take-picture camera (link1 leg))
 (take-picture camera (link2 leg))
 (take-picture camera (link3 leg)))

(defmethod move-incremental-leg ((leg aqua-leg) increment-list k)
 (rotate-link (link0 leg) (leg-attachment-angle leg) k)
 (rotate-link (link1 leg)
        (+ (first increment-list) (inboard-joint-angle (link1 leg))) k)
 (rotate-link (link2 leg)
        (+ (second increment-list) (inboard-joint-angle (link2 leg))) k)
 (rotate-link (link3 leg)
        (+ (third increment-list) (inboard-joint-angle (link3 leg))) k)
 (setf (previous-foot-position leg) (current-foot-position leg))
 (setf (current-foot-position leg)
     (firstn 3 (first (transformed-node-list (link3 leg)))))
 (setf (motion-complete-flag leg) (not (or (motion-limit-flag (link1 leg))
      (motion-limit-flag (link2 leg)) (motion-limit-flag (link3 leg))))))

(defmethod feasible-movep ((leg aqua-leg) allowable-sinkage allowable-slippage)
 (and (<= (third (current-foot-position leg)) allowable-sinkage)
    (or (minusp (third (current-foot-position leg)))
      (minusp (third (previous-foot-position leg)))
      (<= (vector-magnitude (vector-slippage leg)) allowable-slippage))))

(defmethod vector-slippage ((leg aqua-leg))
 (vector-subtract (rest (reverse (previous-foot-position leg)))
          (rest (reverse (current-foot-position leg)))))
```

I. AQUA-LINK.CL

```
;*****************************************************
;Aqua-link.cl defines the class aqua-link associated with the
;     Aquarobot kinematic model.
;This code was written by Prof. McGhee and modified by the author.
;     All modifications are annotated with **.
;*****************************************************


(defclass link0 (rotary-link)
  ((twist-angle :initform 0)
   (link-length :initform 37.5)
   (inboard-joint-angle :initform 0)
   (inboard-joint-displacement :initform 0)
   (min-joint-angle :initform (deg-to-rad -360))
   (max-joint-angle :initform (deg-to-rad 360))
   (node-list :initform '((0 0 0 1) (0 0 0 1) (-37.5 0 0 1)))
   (polygon-list :initform '((1 2)))))


(defclass link1 (rotary-link)
  ((twist-angle :initform (deg-to-rad -90))
   (link-length :initform 20)
   (inboard-joint-angle :initform 0)
   (inboard-joint-displacement :initform 0)
   (min-joint-angle :initform (deg-to-rad -60))
   (max-joint-angle :initform (deg-to-rad 60))
   (node-list :initform '((0 0 0 1) (0 0 0 1) (-20 0 0 1)))
   (polygon-list :initform '((1 2)))))


(defclass link2 (rotary-link)
  ((twist-angle :initform 0)
   (link-length :initform 50)
   (inboard-joint-angle :initform (deg-to-rad 66.4))
   (inboard-joint-displacement :initform 0)
   (min-joint-angle :initform (deg-to-rad -106.6))
   (max-joint-angle :initform (deg-to-rad 73.4))
   (node-list :initform '((0 0 0 1) (0 0 0 1) (-50 0 0 1)))
   (polygon-list :initform '((1 2)))))


(defclass link3 (rotary-link)
  ((twist-angle :initform 0)
   (link-length :initform 100)
   (inboard-joint-angle :initform (deg-to-rad -156.4))
   (inboard-joint-displacement :initform 0)
   (min-joint-angle :initform (deg-to-rad -156.4))
   (max-joint-angle :initform (deg-to-rad 23.6))
   (node-list :initform '((0 0 0 1) (0 0 0 1) (-100 0 0 1)))
   (polygon-list :initform '((1 2)))))


(defmethod update-A-matrix ((link link))
  (with-slots (twist-angle link-length inboard-joint-angle
     inboard-joint-displacement A-matrix) link
     (setf A-matrix (dh-matrix (cos inboard-joint-angle)
       (sin inboard-joint-angle) (cos twist-angle) (sin twist-angle)
       link-length inboard-joint-displacement))))
```

```
(defmethod rotate ((link rotary-link) angle)
 (setf (inboard-joint-angle link) angle)
 (update-A-matrix link)
 (setf (H-matrix link) (matrix-multiply (H-matrix (inboard-link link))
                                        (A-matrix link)))
 (transform-node-list link))


(defmethod rotate-link ((link rotary-link) angle k)
 (cond ((> angle (- (max-joint-angle link) (deg-to-rad k)))
                          (rotate link (max-joint-angle link))
                     (setf (motion-limit-flag link) t))
      ((< angle (+ (min-joint-angle link) (deg-to-rad k)))
                          (rotate link (min-joint-angle link))
                  (setf (motion-limit-flag link) t))
      (t (rotate link angle) (setf (motion-limit-flag link) nil))))


(defmethod calc-kinematic-margin-link ((link rotary-link) angle k);;**
  (- (min (rad-to-deg (- angle (min-joint-angle link)))
       (rad-to-deg (- (max-joint-angle link) angle)))
     k))
```

J. AQUA-INV-KINEMATICS.CL

```
;*******************************************************
;Aqua-inv-kinematics.cl defines the inverse kinematics associated with the
;     Aquarobot kinematic model.
;This code was written by [Ref. 9] and modified by the author.
;     All modifications are annotated with **.
;*******************************************************


(defconstant L2sqr (sqr 50.0))
(defconstant L3sqr (sqr 100.0))

; assumptions: dh coord system for link0 of respective leg:
;           origin at joint1,
;           x-axis directed away from center of body,
;           z-axis aligned with body z-axis;
;         foot-position = '(x y z).
(defun theta1 (foot-position)
    (atan (cadr foot-position)(car foot-position)));;**


; assumptions: dh coord system for link1 of respective leg:
;           origin at joint2,
;           x-axis directed away from joint1,
;           z-axis aligned with body z-axis;
;         foot-position = '(x y z);
;         hyp = distance from joint2 to foot.
(defun theta2 (foot-position hyp hyp-sqr leg)
  (cond ((and (< hyp 150.0);if hyp is shorter than the length of the leg;**
          (> hyp 50.0));;**
       (- (acos (/ (+ L2sqr hyp-sqr (- L3sqr)) (* 2 50.0 hyp)))
       (if (< (car foot-position) 0)
           (- pi (asin (/ (caddr foot-position) hyp)))
         (asin (/ (caddr foot-position) hyp)))))
      (t (cond ((>= hyp 150.0);;**
               (- (min-joint-angle (link2 leg)) 0.01745));;**
         (t (+ (max-joint-angle (link2 leg)) 0.01745))))));;**


; assumptions: same as for theta2.
(defun theta3 (foot-position hyp-sqr hyp leg)
  (cond ((and (< hyp 150.0);;**
          (> hyp 50.0));;**
       (- (acos (/ (+ L2sqr L3sqr (- hyp-sqr)) (* 2 50.0 100.0))) pi))
      (t (cond ((>= hyp 150.0) ;;**
               (+ (max-joint-angle (link3 leg)) 0.01745));;**
         (t (- (min-joint-angle (link3 leg)) 0.01745))))));;**


; returns foot position with respect to joint 1 in link0 coord.
(defmethod foot-joint1/link0coord ((leg aqua-leg) foot-pos)
   (world-to-body (link0 leg) foot-pos));**
```

```
; returns foot position with respect to joint 2 in link1 coord.
; given foot-joint1/link0coord.
(defun foot-joint2/link1coord (foot-pos)
 (list (- (sqrt (+ (sqr (car foot-pos)) (sqr (cadr foot-pos)))) 20)
      0 (caddr foot-pos)))


; returns list of joint angles required for given (world coord) foot position.
(defmethod aqua-inv-kin ((leg aqua-leg) foot-position)
  (let* ((pos0 (foot-joint1/link0coord leg foot-position))
        (pos1 (foot-joint2/link1coord pos0))
        (hyp-sqr (+ (sqr (car pos1)) (sqr (caddr pos1))))
        (hyp (sqrt hyp-sqr)))
       (list (theta1 pos0)
            (theta2 pos1 hyp hyp-sqr leg)
            (theta3 pos1 hyp-sqr hyp leg))))
```

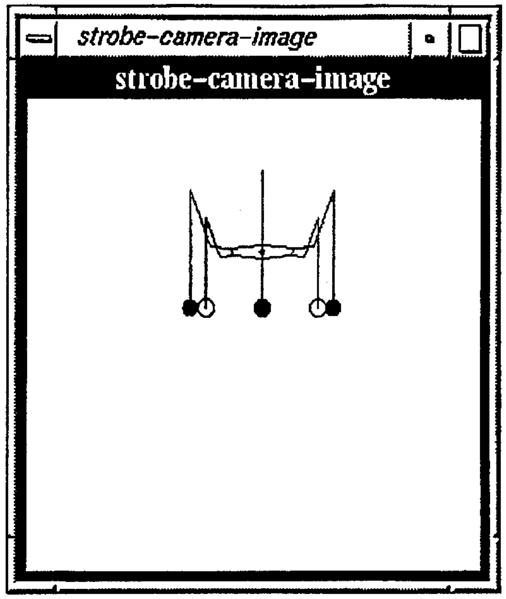
APPENDIX B. SAMPLE SIMULATIONS

This appendix constains sample simulations of kinematic and stability margins.

A. JOINT SPACE MOTION MARGINS

```
Allegro CL 4.1 [SPARC; R1] (2/9/94 14:30)

;; Copyright Franz Inc., Berkeley, CA, USA
;; Unpublished. All rights reserved under the copyright laws
;; of the United States.

;; Restricted Rights Legend
;; ------------------------
;; Use, duplication, and disclosure by the Government are subject to
;; restrictions of Restricted Rights for Commercial Software developed
;; at private expense as specified in DOD FAR 52.227-7013 (c) (1) (ii).
;; Optimization settings: safety 1, space 1, speed 1, debug 2
;; For a complete description of all compiler switches given the current
;; optimization settings evaluate (explain-compiler-settings).


user(1): (load "load-files.cl")
; Loading /users/work1/dunton/cs4314/project/load-files.cl.
t


user(2): (load-aqua)
; Loading /users/work1/dunton/cs4314/project/strobe-camera.cl.
; Loading /users/work1/dunton/cs4314/project/misc.cl.
; Loading /users/work1/dunton/cs4314/project/link.cl.
; Loading /users/work1/dunton/cs4314/project/rigid-body.cl.
; Loading /users/work1/dunton/cs4314/project/robot-kinematics.cl.
; Loading /users/work1/dunton/cs4314/project/aqua.cl.
; Loading /users/work1/dunton/cs4314/project/aqua-leg.cl.
; Loading /users/work1/dunton/cs4314/project/aqua-link.cl.
; Loading /users/work1/dunton/cs4314/project/aqua-inv-kinematics.cl.
t
```

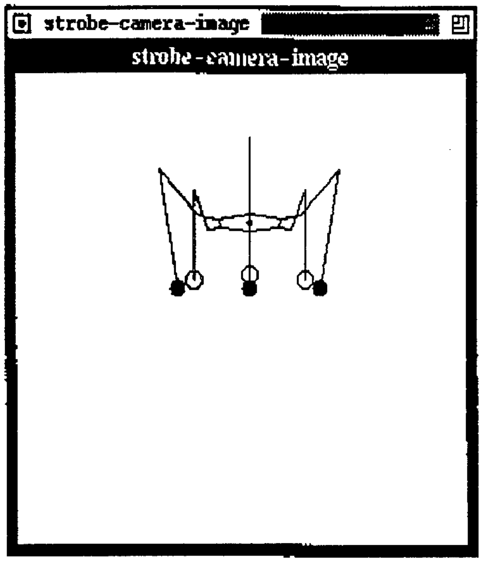
user(3): (aqua-picture)



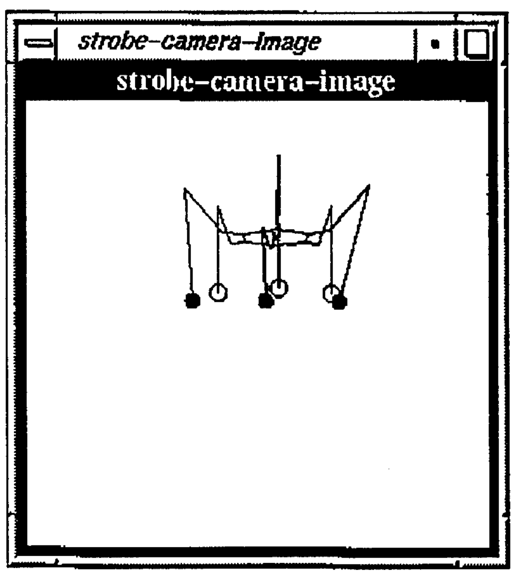

```
*********************************************************************
  --In order to move aquarobot, use the following key/mouse
    combinations:

        Translation                    Rotation

     left-click      =     +x   =  CNTRL+left-click
     middle-click  =     +y   =  CNTRL+middle-click
     right-click     =     +z   =  CNTRL+right-click

     SHFT+left-click      =     -x
     SHFT+middle-click  =     -y
     SHFT+right-click     =     -z

   Alternatively, movement may be specified as:

     (change-posture-incremental aqua increment-list)

   The only allowable first move is in the -z direction.  All
   other movements will result in an error message.

  --To specity kinematic and stability safety margins, enter
    enter the following:

     (specify-safety-margins aqua k s)

   k is in degrees and s is in cm. The default values are 0.

*********************************************************************
nil
```

user(4): In mouse-handler button: 48

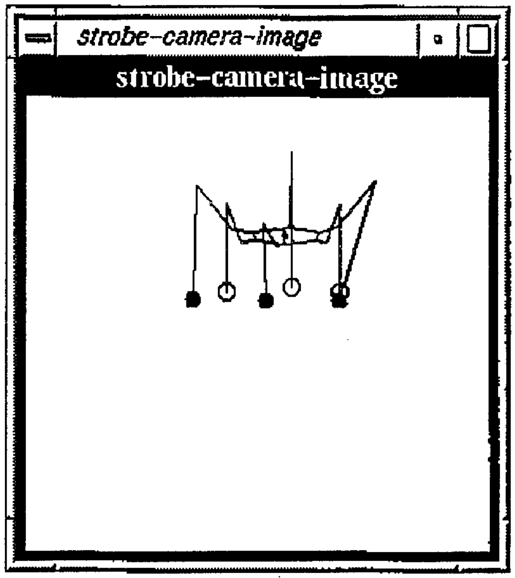
Mouse Event: SHFT+Right-click: translate-z
Joint Space Motion margin = 13.4606 degrees
Stability margin = 38.7587 cm



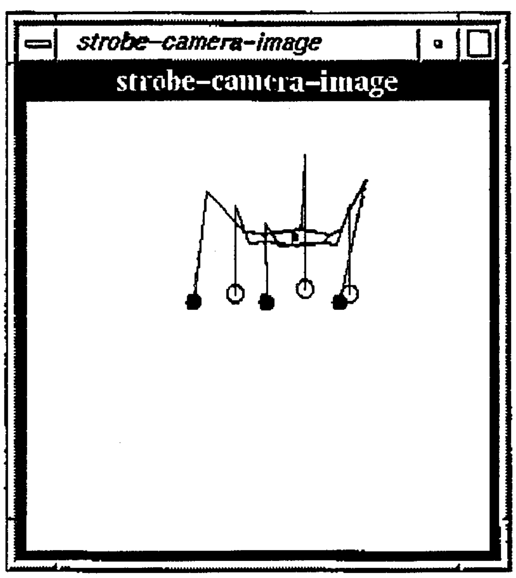
In mouse-handler button: 64
Mouse Event: Middle-click: translate+y
Joint Space Motion margin = 10.8870 degrees
Stability margin = 30.0985 cm



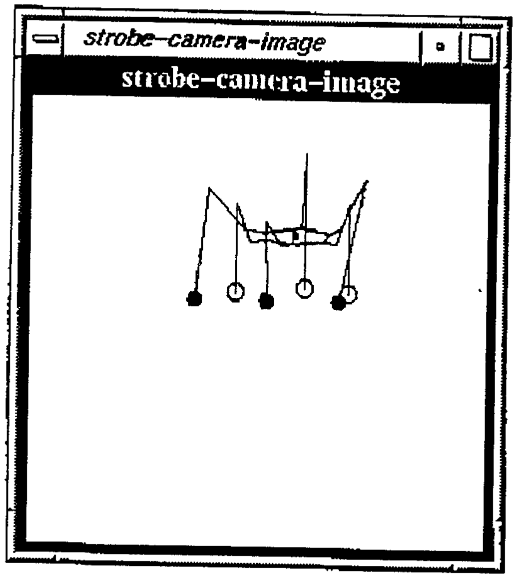
In mouse-handler button: 64
Mouse Event: Middle-click: translate+y
Joint Space Motion margin = 9.7047 degrees
Stability margin = 21.4382 cm



In mouse-handler button: 64
Mouse Event: Middle-click: translate+y
Joint Space Motion margin = 9.4663 degrees
Stability margin = 12.7780 cm



In mouse-handler button: 64
Mouse Event: Middle-click: translate+y
Error: Joint limit reached in leg3: Joint 1
Stability margin = 4.1177 cm

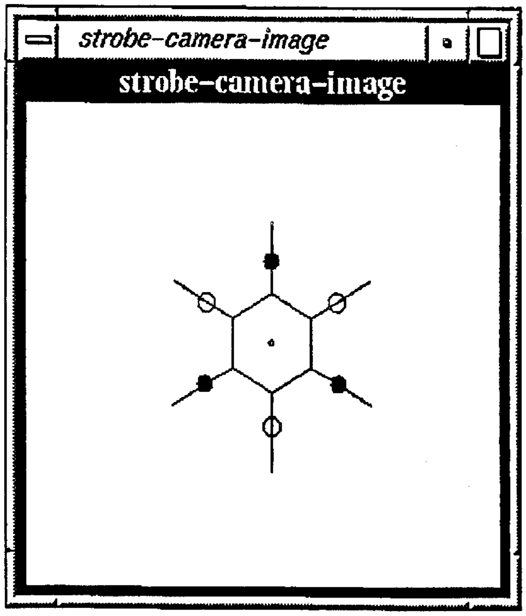


user(4): (exit)
; killing "Default Window Stream Event Handler"
; killing "X11 event dispatcher"
; killing "Initial Lisp Listener"
; Exiting Lisp

B. STABILITY MARGINS

```
Allegro CL 4.1 [SPARC; R1] (2/9/94 14:30)

;; Copyright Franz Inc., Berkeley, CA, USA
;; Unpublished. All rights reserved under the copyright laws
;; of the United States.

;; Restricted Rights Legend
;; -----------------------
;; Use, duplication, and disclosure by the Government are subject to
;; restrictions of Restricted Rights for Commercial Software developed
;; at private expense as specified in DOD FAR 52.227-7013 (c) (1) (ii).
;; Optimization settings: safety 1, space 1, speed 1, debug 2
;; For a complete description of all compiler switches given the current
;; optimization settings evaluate (explain-compiler-settings).


user(1): (load "load-files.cl")
; Loading /users/work1/dunton/cs4314/project/load-files.cl.
t


user(2): (load-aqua)
; Loading /users/work1/dunton/cs4314/project/strobe-camera.cl.
; Loading /users/work1/dunton/cs4314/project/misc.cl.
; Loading /users/work1/dunton/cs4314/project/link.cl.
; Loading /users/work1/dunton/cs4314/project/rigid-body.cl.
; Loading /users/work1/dunton/cs4314/project/robot-kinematics.cl.
; Loading /users/work1/dunton/cs4314/project/aqua.cl.
; Loading /users/work1/dunton/cs4314/project/aqua-leg.cl.
; Loading /users/work1/dunton/cs4314/project/aqua-link.cl.
; Loading /users/work1/dunton/cs4314/project/aqua-inv-kinematics.cl.
t
```

user(3): (aqua-picture)



****
--In order to move aquarobot, use the following key/mouse combinations:

Translation				Rotation
left-click	=	+x	=	CNTRL+left-click
middle-click	=	+y	=	CNTRL+middle-click
right-click	=	+z	=	CNTRL+right-click

SHFT+left-click	=	-x
SHFT+middle-click	=	-y
SHFT+right-click	=	-z

Alternatively, movement may be specified as:

(change-posture-incremental aqua increment-list)

The only allowable first move is in the -z direction. All other movements will result in an error message.

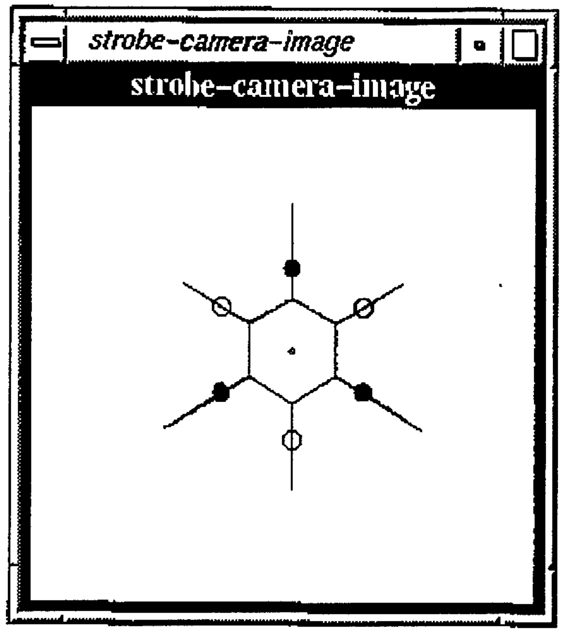
--To specity kinematic and stability safety margins, enter enter the following:

(specify-safety-margins aqua k s)

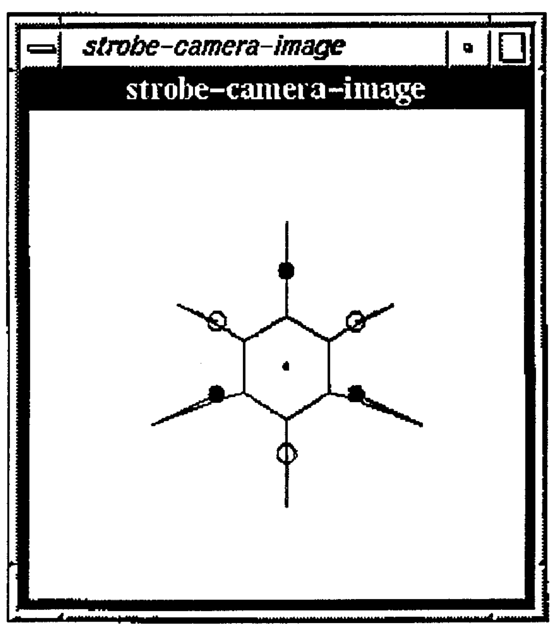
k is in degrees and s is in cm. The default values are 0.

****
nil

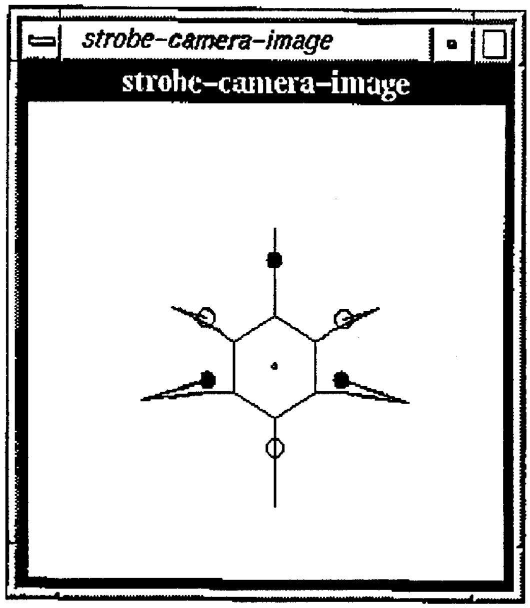
user(4): In mouse-handler button: 48
Mouse Event: SHFT+Right-click: translate-z
Joint Space Motion margin = 13.4606 degrees
Stability margin = 38.7587 cm



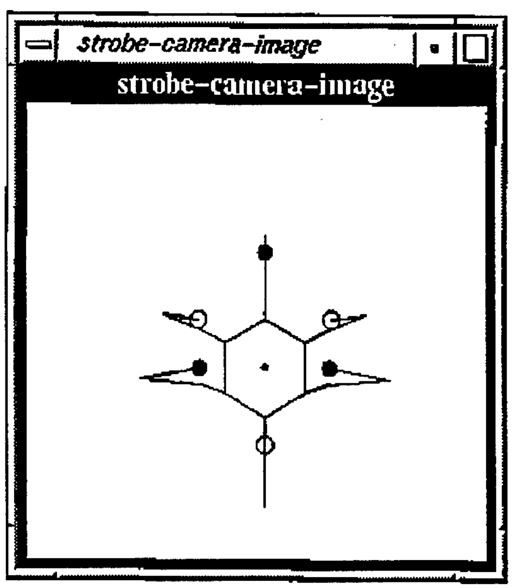
In mouse-handler button: 144
Mouse Event: SHFT+Left-click: translate-x
Joint Space Motion margin = 12.0850 degrees
Stability margin = 28.7587 cm



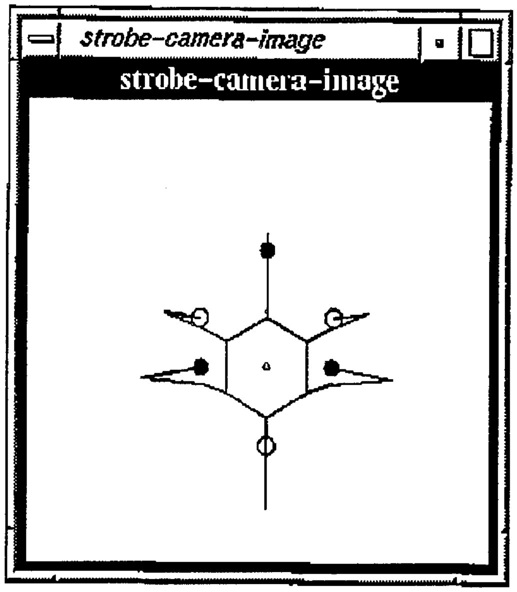
In mouse-handler button: 144
Mouse Event: SHFT+Left-click: translate-x
Joint Space Motion margin = 11.6553 degrees
Stability margin = 18.7587 cm



In mouse-handler button: 144
Mouse Event: SHFT+Left-click: translate-x
Joint Space Motion margin = 12.0835 degrees
Stability margin = 8.7587 cm



In mouse-handler button: 144
Mouse Event: SHFT+Left-click: translate-x
Joint Space Motion margin = 0.0217 degrees
Error: Stability margin reached: Tripod segment 35



user(4): (exit)
; killing "Default Window Stream Event Handler"
; killing "X11 event dispatcher"
; killing "Initial Lisp Listener"
; Exiting Lisp

LIST OF REFERENCES

1. McGhee, R., "Vehicular Legged Locomotion," *Advances in Automation and Robotics,* Volume 1, edited by G.N. Saridis, JAI Press Inc., pp. 259-284,1985.

2. Goetz, J., *Graphical Simulation of Articulated Ridgid Body System Kinematics with Collision Detection,* Master's Thesis, Naval Postgraduate School, Monterey, California, March, 1994.

3. Kwak, S. H. and McGhee, R., *Rule-Based Motion Coordination for the Adaptive Suspension Vehicle*, Technical Report No. NPS52-88-011, Naval Postgraduate School, Monterey, California, May, 1988.

4. Raibert, M., *Legged Robots that Balance,* MIT Press, Cambridge, Massachusetts, 1986.

5. Kwak, S. H. and McGhee, R., "Rule-Based Motion Coordination for a Hexapod Walking Machine," *Advanced Robotics*, July, 1989, Volume 4, Number 3, pp. 263-282.

6. McGhee, R., and Iswandhi, G., "Adaptive Locomotion of a Multilegged Robot over Rough Terrain," *IEEE Transactions on Systems, Man and Cybernetics*, April, 1979, Volume 9, Number 4, pp. 176-182.

7. Kanayama, Y., et al., "An International Joint Research Project on an Autonomous Underwater Walking Robot," *Proceedings of the International Symposium on Coastal Ocean Space Utilization*, Yakohama, Japan, May 30 - June 2, 1995.

8. Davidson, S., *An Experimental Comparison of CLOS and C++ Implementations of an Object-Oriented Graphical Simulation of Robot Kinematics*, Master's Thesis, Naval Postgraduate School, Monterey, California, March, 1993.

9. Kristiansen, K., *A Computer Simulation of Vehicle and Actuator Dynamics for a Hëxapod Walking Robot*, Master's Thesis, Naval Postgraduate School, Monterey, California, March, 1994.

10. McMillan, S., *Computational Dynamics for Robotic Systems on Land and Under Water*, Ph.D. Dissertation, Ohio State University, Columbus, Ohio, September, 1994.

11. Schue, C., *Gait Planning for a Hexapod Underwater Walking Machine,* Master's Thesis, Naval Postgraduate School, Monterey, California, June, 1993.

12. Yoneda, K., Suzuki, K., and Kanayama, Y., "Gait Planning for Versatile Motion of a Six Legged Robot," *Proceedings of IEEE International Conference on Robotics and Automation,* San Diego, California, May, 1994.

13. Craig, J., *Introduction to Robotics: Mechanics and Control,* Second Edition, Addison-Wesley Publishing Company, Inc., Menlo Park, California, 1989.

14. Frank, A. and McGhee, R., "Some Considerations Relating to the Design of Autopilots for Legged Vehicles," *Journal of Terramechanics*, 1969, Volume 6, Number 1, pp. 23 - 35.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..2
   Cameron Station
   Alexandria, VA 22304-6145

2. Dudley Knox Library...2
   Code 52
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Chairman, Code CS..2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

4. Dr Robert McGhee, Code CS/Mz...2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

5. Dr Michael Zyda, Code CS/Zk...2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

6. Dr Yutaka Kanayama, Code CS/KA...1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

7. Dr Se-Hung Kwak 1
   Loral/ADS
   50 Moulton St.
   Cambridge, MA 02138

8. Mr Norman Caplan..1
   Biological and Critical Systems Division
   Engineering Directorate
   National Science Foundation
   4201 Wilson Blvd
   Arlington, VA 22230

9. Ms Shirley Pratt, Code CS..1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

10. Lt Elizabeth M. Dunton..1
7656 Lexington Manor Dr
Colorado Springs, CO 80920